

**课 程 实 验 报 告**

**课程名称： 计算机系统基础实验**

**实验名称： 数据的表示**

**院 系 ：计算机科学与技术**

**专业班级 ： cs2208**

**学 号 ： U202215642**

**姓 名 ： 田清林**

**指导教师 ： 班鹏新**

**2024 年 3 月 19 日**

**一、实验目的与要求**

⑴ 熟练掌握程序开发平台(VS2019) 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。

1. 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;}

1. int negate(int x); 不使用负号，实现 -x

判断函数： int netgate\_standard(int x) { return -x;}

1. 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;}

1. int bitOr(int x, int y); 仅使用 ~ 和 &，实现 |
2. int bitXor(int x, int y); 仅使用 ~ 和 &，实现 ^
3. int isTmax(int x); 判断x是否为最大的正整数（7FFFFFFF），

只能使用 !、 ~、 &、 ^、 |、 +

1. 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

1. 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。
2. 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

1. 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：

分析结构体各成员，可以发现两个char数组中存在声明空间过大的问题，用于存储学生年龄的short成员表示范围过大，而float为了防止精度丢失的问题暂不进行压缩，除此之外，还存在由于成员对齐引入的冗余空间（short和float之间有两字节的冗余）。

压缩策略：

1.压缩时按照结构体各成员声明的顺序进行处理，即name[8],age,score,remark[200]。

2.对于name和remark，按字节存储非0字符直到字符串结尾(‘\0’),存入‘\0’作为分隔符，之后不再存储字符。

3.对于short,只存储其低字节(本机采用小端法存储，为低地址位置)，舍去高字节。

4.对于float，4个字节按地址从低往高顺序依次存储。

编写逐字节压缩函数int pack\_student\_bytebybyte(student\* s, int sno, char \*buf)，只需根据上方规则对每个字节进行操作即可。

对于一条语句整体写入函数int pack\_student\_whole(student\* s, int sno, char \*buf)，name和remark可以使用strcpy实现同样效果，short仅存储1字节可用char指针一步写入，float可以使用memcpy函数将4个字节一次性复制到压缩数组中。

对于解压缩函数int restore\_student(char \*buf, int len, student\* s)，只需按照成员声明的顺序反向操作即可。

运行结果：

首先输入5个学生的信息，如图1红色方框所示。

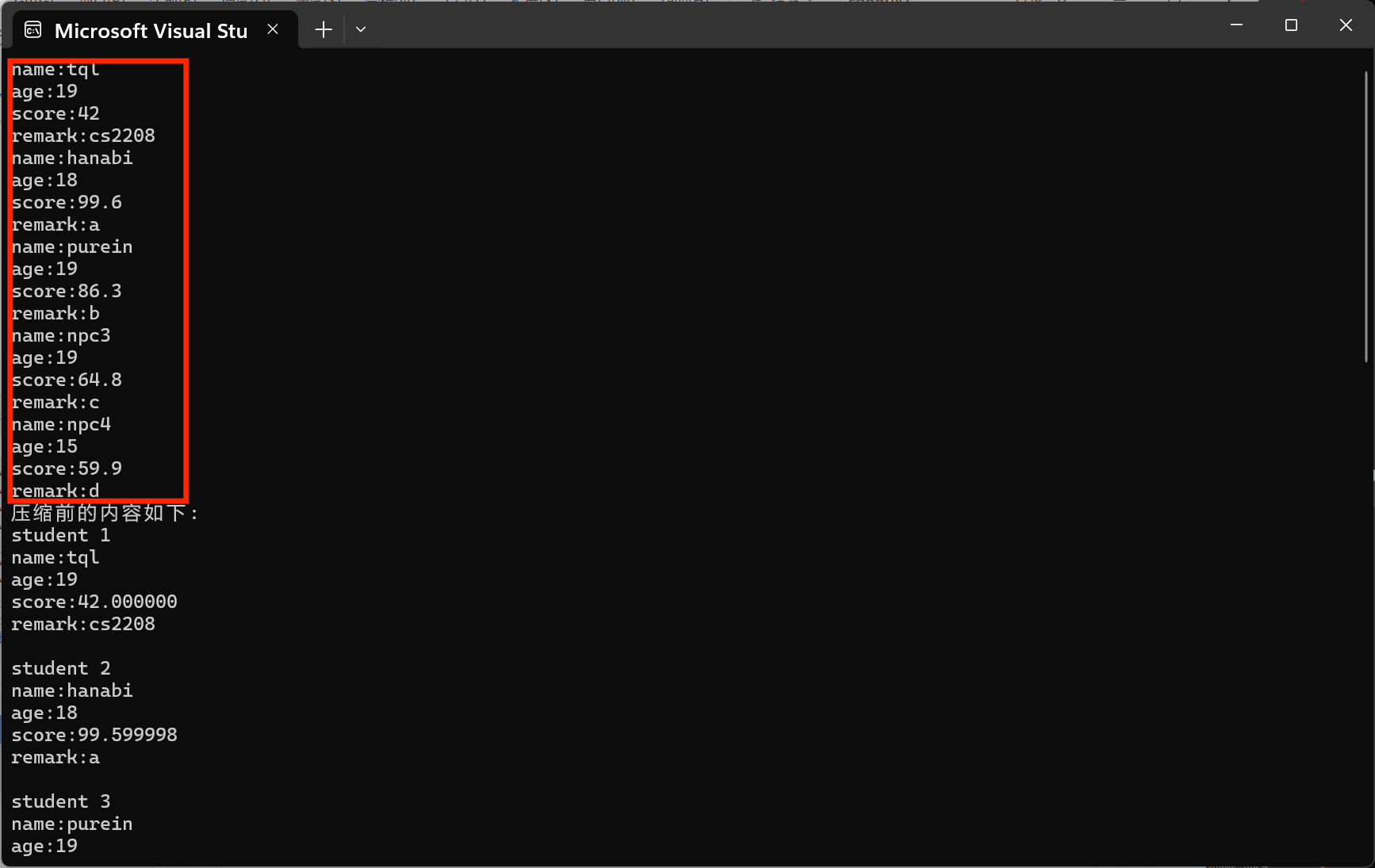


图 1 输入学生信息

接下来输出压缩前的内容，如图2红色方框所示。

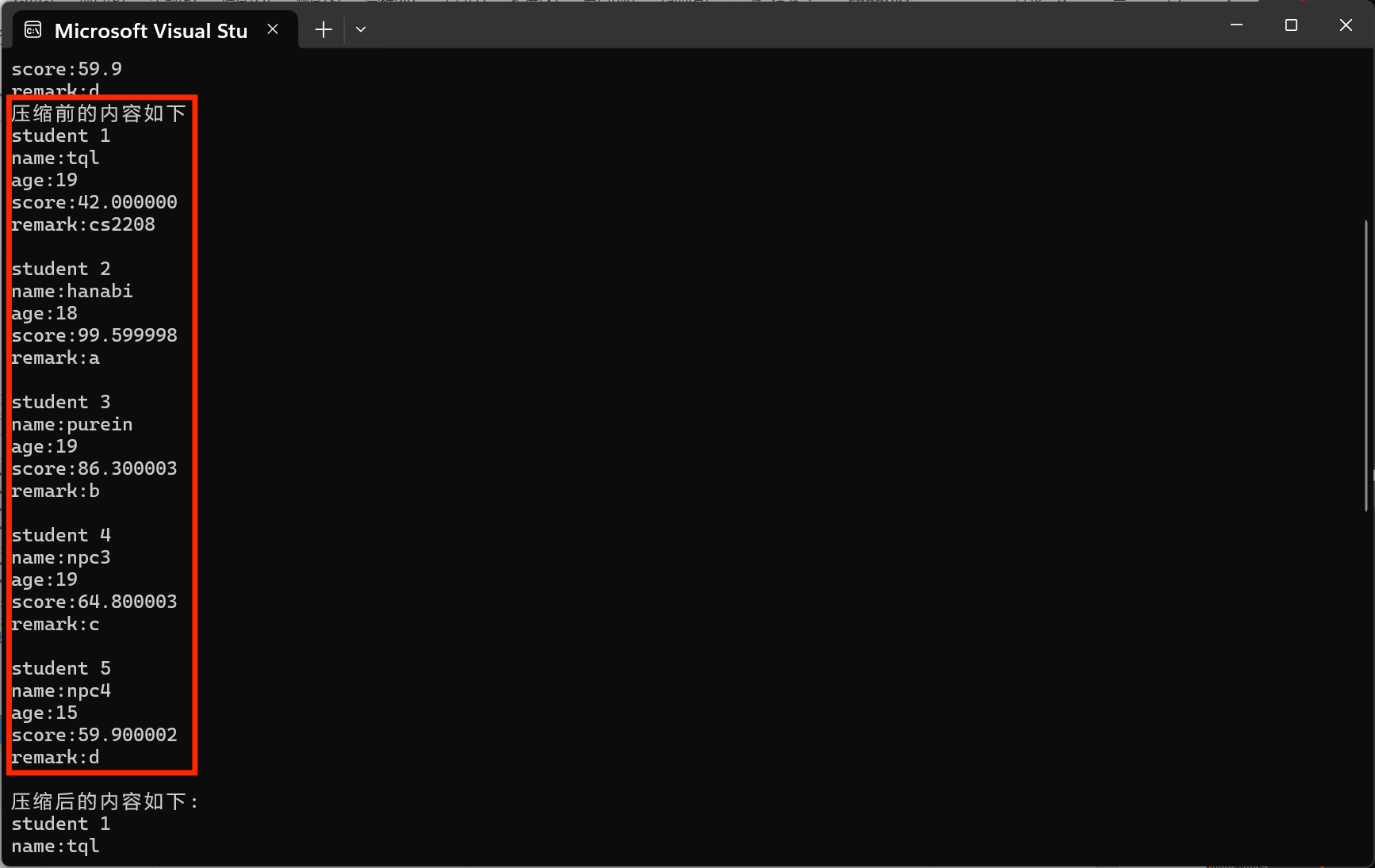


图 2压缩前的内容

输出压缩后解压出的内容与压缩前后存放数据长度，分别为1080和64，如图3所示。

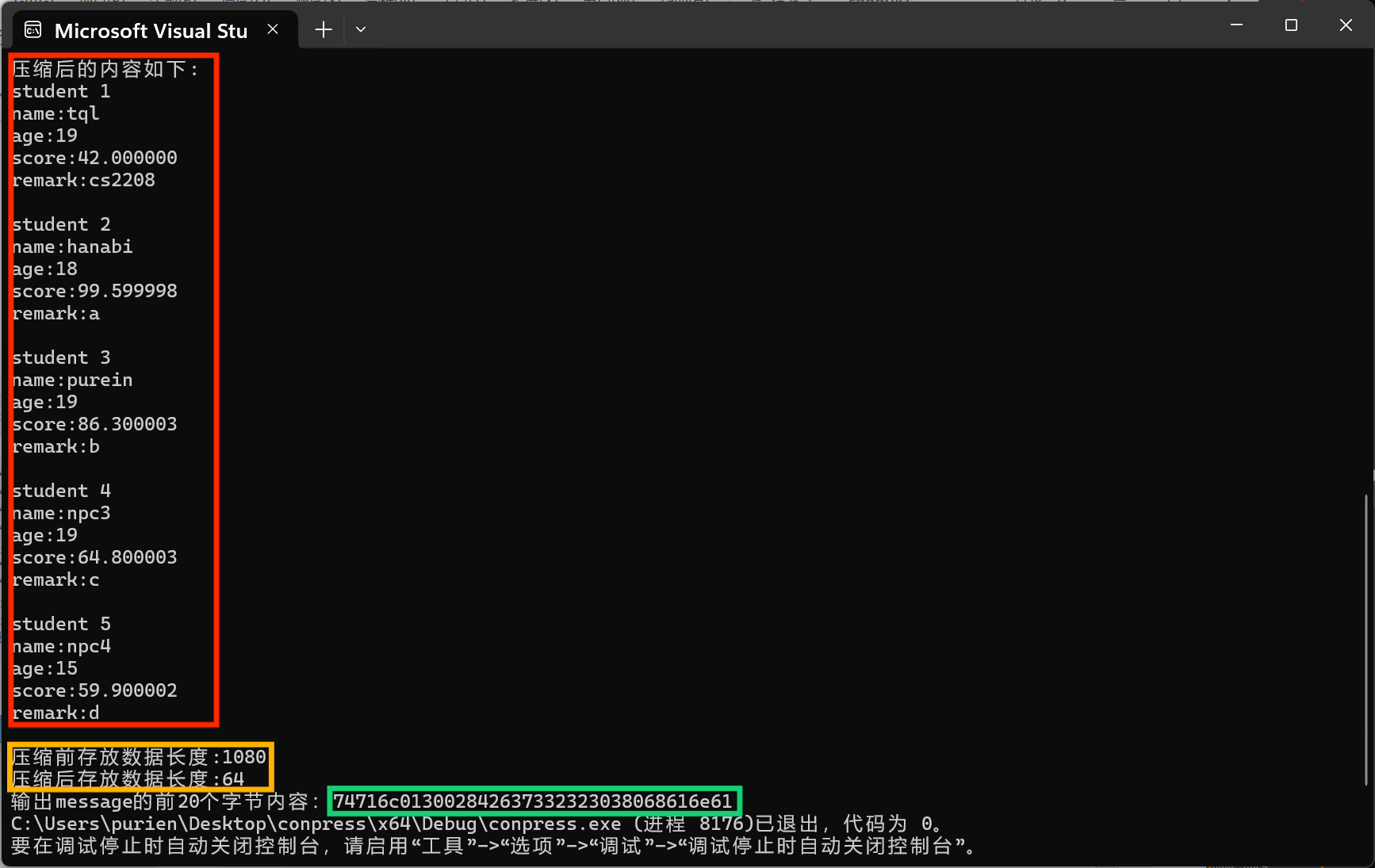


图 3 压缩后的内容，压缩前后存放数据长度和message前20个字节内容

从监视窗口中得到message的首地址和new\_s[0].score的首地址,并在内存窗口中输入地址进行查看。监视窗口内容如图4所示。

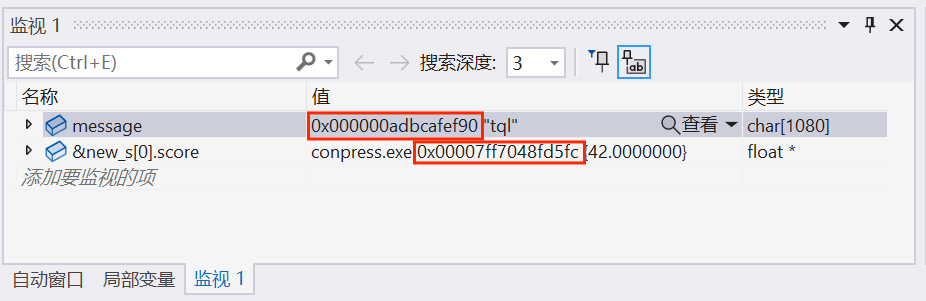


图 4 监视窗口

打开内存窗口查看message前20个字节内容，与输出的message前20个字节内容（如图3绿色方框所示）相比较，发现二者相同，符合预期。

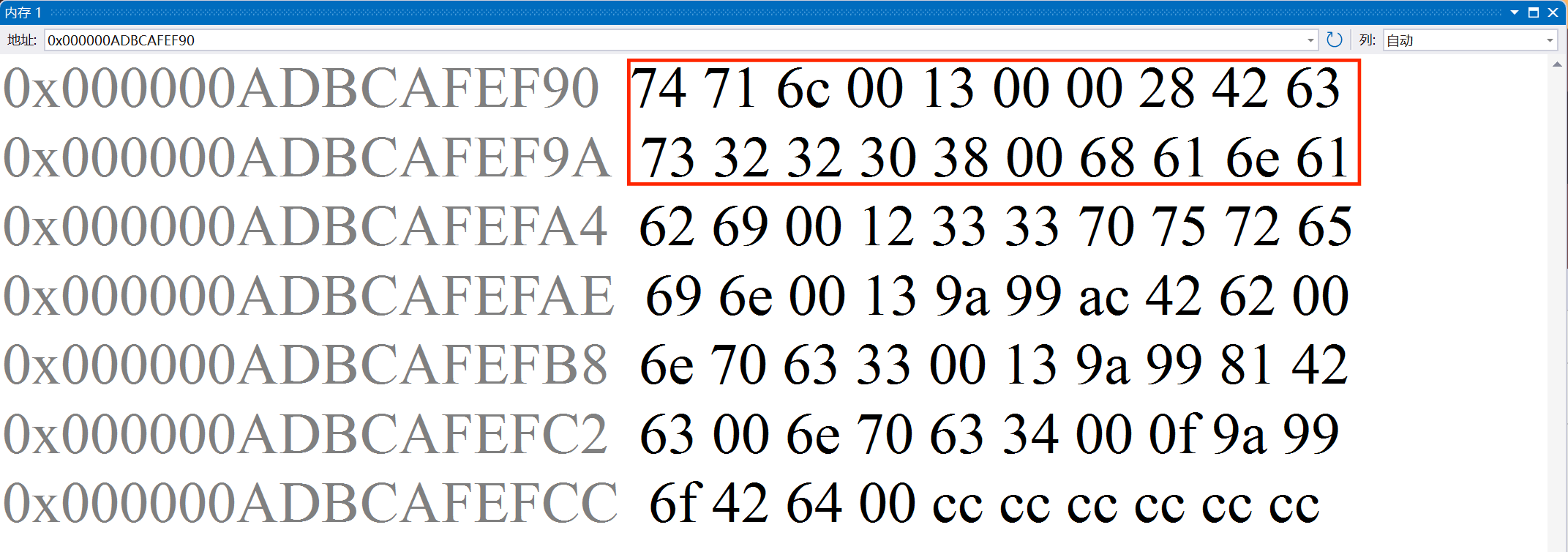


图 5内存窗口message内容

找到第0个学生的score的地址并在内存窗口中查看，取出前4个字节（float在本机上占4个字节）,按从低地址到高地址的字节顺序为 00 00 28 42。

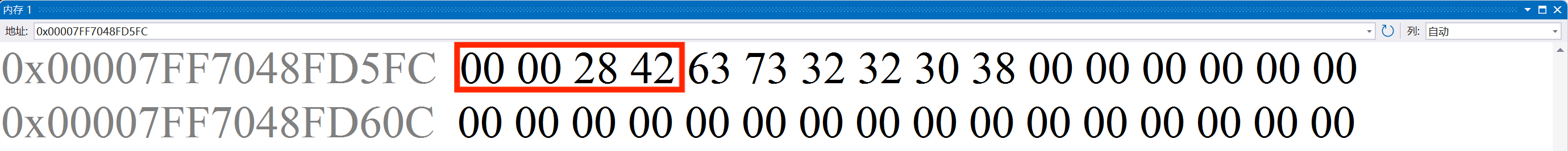


图 6内存窗口score内容

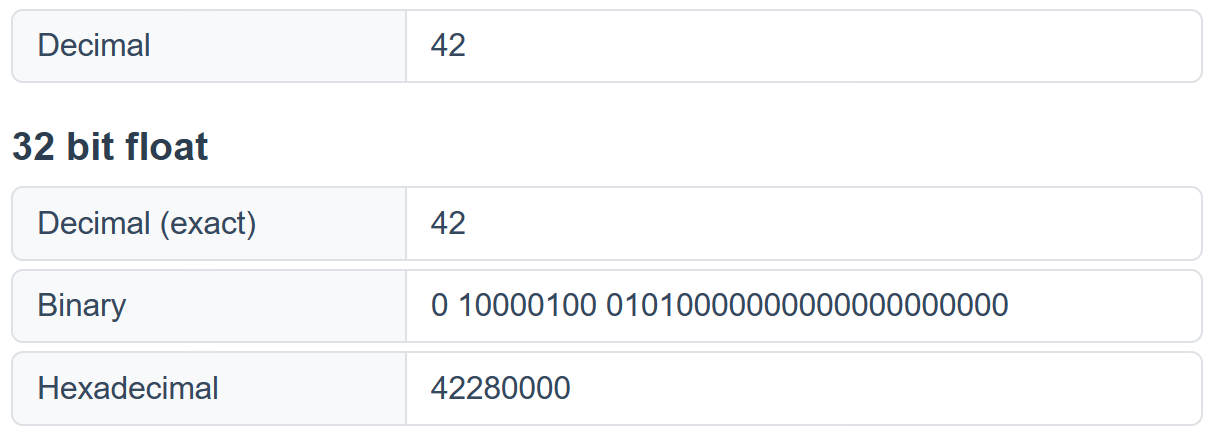


图 7 浮点数42的表示

浮点数的表示遵循IEEE754标准，1位符号位S，8位阶码E，23位尾数M,根据浮点数42的二进制表示，此处符号位S=0，阶码E = 0b1000 0100 = 132 = 127 + 5，M = 0b010 1000 0000 0000 0000 0000。

浮点数42的16进制表示为 42 28 00 00，正好与本机内存中字节顺序相反，可以看出本机采用小端法存储数据。

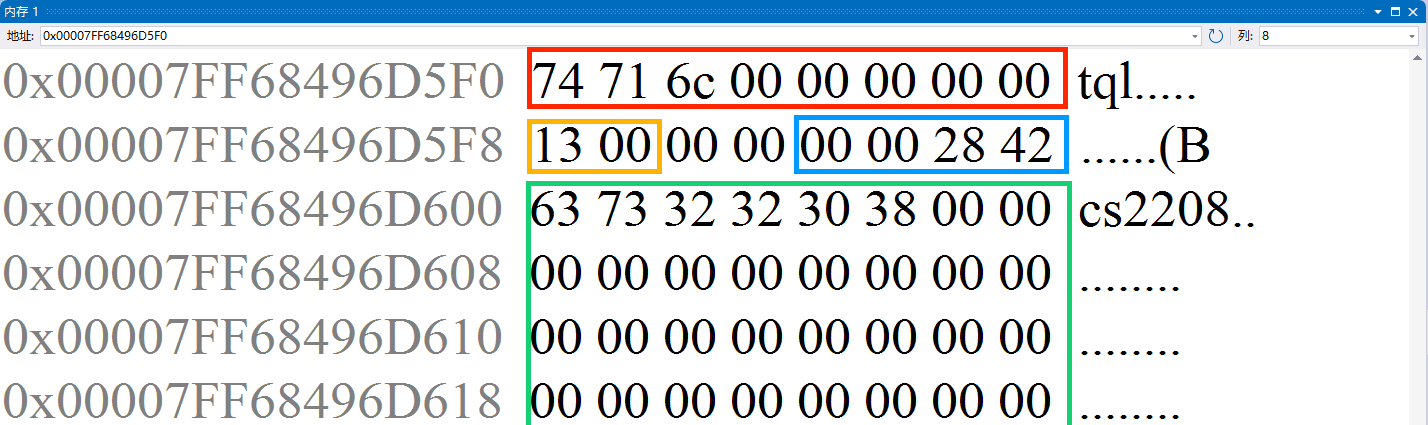


图 8 内存中结构数组的存储

观察内存中结构数组的成员的存放,对于student[0]的各成员内存占用如上所示，char name[8],short age, float score, char remark[200]依次分别占用红色，黄色，蓝色，绿色区域。可以发现：

1.各成员在内存中存储顺序与成员声明顺序相同

2.不同类型成员遵循不同的对齐机制，即首地址必须是其字节数的整数倍，对于数组元素，视作数组大小个数的原子类型成员的放置。

**（2）任务 2 的算法思想、运行结果等记录**

算法思想：

1. int absVal(int x); 返回 x 的绝对值

仅使用 !、 ~、 &、 ^、 |、 +、 <<、 >>， 运算次数不超过 10次

求x的绝对值即在x大于等于0时返回x，小于0时返回~x+1(即-x),两种情况的区别是符号位，将x算术右移31位根据符号位的不同可以得到全0或全1的二进制串，若将其与x进行异或可以表示两种情况下的x正确选择是否取反，并利用全1的二进制串加1后得到0的特点，稍加组合得到答案。

1. int negate(int x); 不使用负号，实现 -x

有符号整数机内使用补码表示，其负号在位级上可以用取反加一实现。

1. int bitAnd(int x, int y); 仅使用 ~ 和 |，实现 &

使用德·摩根定律，x&y = ~(~x|~y)，可用或非逻辑实现与逻辑。

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

使用德·摩根定律，x|y = ~(~x&~y)，可用与非逻辑实现或逻辑。

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

根据真值表，x^y = x&~y|~x&y,再用与非逻辑实现或逻辑即可。

1. int isTmax(int x); 判断x是否为最大的正整数（7FFFFFFF），

只能使用 !、 ~、 &、 ^、 |、 +

Tmax和-1都具有一个特殊的性质，即x+1 = ~x,只要满足该性质且排除-1，剩下的值就是Tmax，用异或和逻辑非完成比较操作即可判断。

1. 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

①首先，将32位每连续四位分为一组进行累加。使用mask1形如0001 0001……的32位掩码。将x分别右移动0，1，2，3位与mask1执行按位与操作后的值进行累加，得到的结果中每连续四位二进制数表示x中对应连续四位的二进制个数（最大为0100，由于最高有效位为0，方便移位）。

②其次，使用mask2形如00001111 00001111……的32位掩码。将x分别向右移动0，4位与mask2执行按位与操作后的值进行累加，得到的结果中每连续八位二进制数表示x中对应连续八位的二进制个数。

③最后把四个八位二进制串加在一起。由于对于int中1的个数最大为32，即0010 0000，用八位即可完整表示，之后可以直接移位将对应位相，无需考虑字节间进位的影响。最后用0xff作为掩码只保留最低八位，即为答案。

注意：根据运算符的优先级顺序加括号，例如加号+的优先级大于按位与&。

1. 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。

利用~0向左移位，构造两个形如11110000类型的数进行异或，这样只保留中间交叉部分的1。注意由于lowbit和highbit是闭区间，需要左移highbit＋1位与lowbit位，由于c语言移位运算会将移位数对其位数取模，当highbit等于31时会出现移动0位而不是32位的情况，所以选择分两步执行，左移highbit后再左移1位。

1. 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

补码加法当且仅当正正得负或负负得正时发生溢出，只需关注操作数和结果的符号位，进行恰当的按位运算后右移31位使得符号位运算的结果填充全部32位，用逻辑非！判断是否为0来判断是否溢出。

1. 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

将结果分为三部分，两个需要交换的字节位和无需改变的其余字节，三部分可用加法+或按位或|进行组合。用0xff进行左移来造出只有第n个字节和m个字节为1，其他全为0的掩码来取出对应位，计算出m与n的差后将目标字节左移/右移到目标位置，其余字节用x与先前两个掩码按位或结果取反得到的掩码取出，三部分组合在一起即为答案。

在自动检测程序中，对于每个题目，用预先设置好的测试数据分别调用标准函数和独自编写函数，分别输出结果并判断结果是否相同。

运行结果：

自动检测程序测试结果如下。

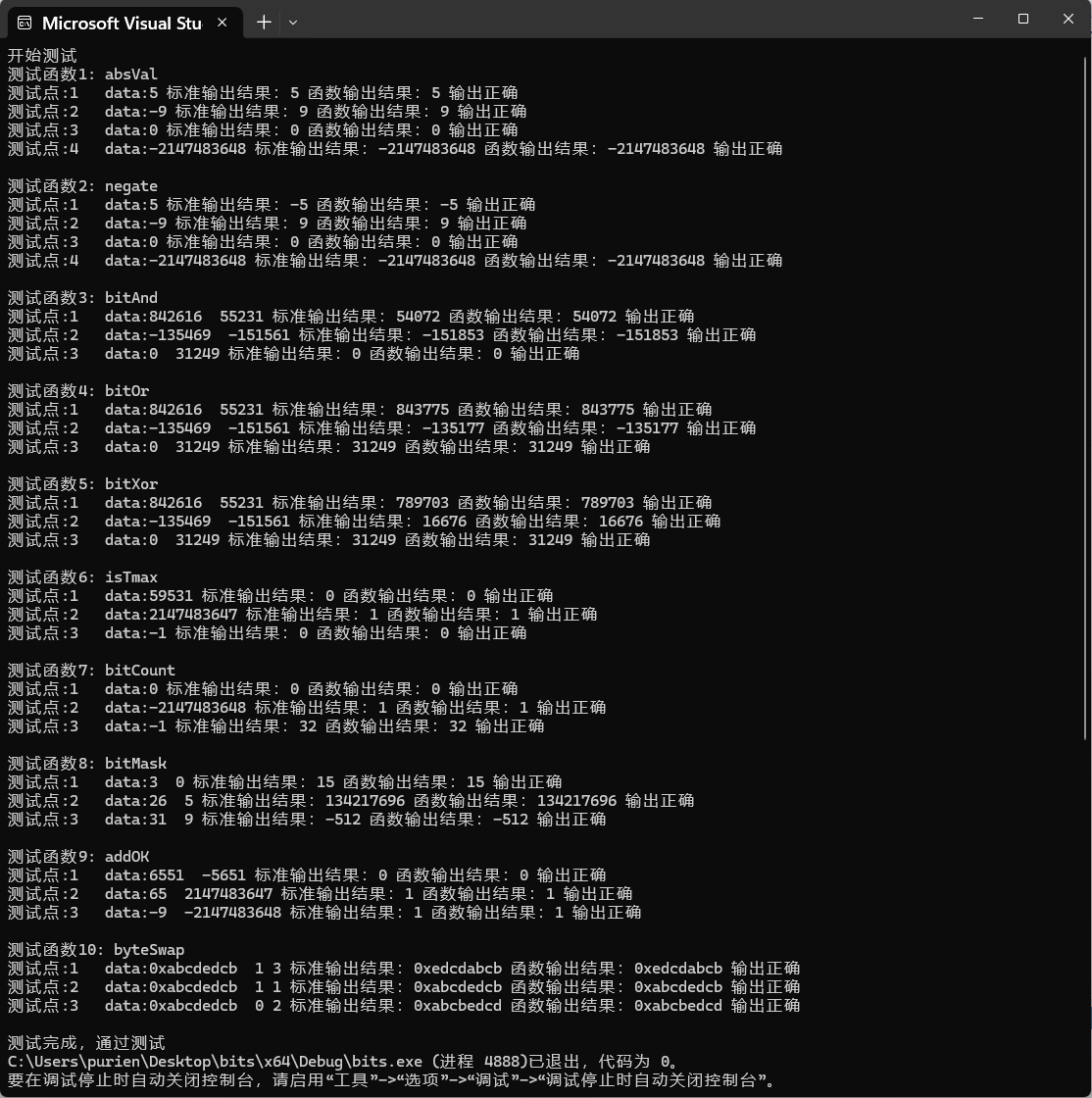


图 9 bits.c测试结果

10个函数全部成功实现，通过测试。

**四、体会**

本次实验作为计算机系统基础的第一次实验，难度不高，但仍让我受益匪浅。

在任务一的压缩过程中，我曾尝试对float进行压缩。深入研究浮点数编码后，我根据成绩的数据范围两位小数的精度对编码进行设计，采取舍弃符号位，4位阶码和20位尾数的表示策略，舍去符号位，默认成绩为正数。这样既能保证阶码的E范围为0到15，引入bias = 7和与 IEEE754标准相同的表示NaN，无穷和非规格化规则，实际表示的规格化最大值为2^(15-1-7)\*(2-2^(-20))约等于259.99，最小值为编码可以保证在压缩前后十进制三位小数相同。但后来由于并没有给出score范围，仅仅用平时的满分100或150来作为标准欠妥当，而如果考虑到高考成绩的750甚至sat成绩的1600，三字节无法兼顾表示范围和表示精度，为此无奈选择放弃（实际上5位阶码与19位尾数即可完成）。不过在思考的过程中加深了我对于浮点数编码的理解与掌握。

在任务二中，我通过一系列的位级操作，加深了对位运算、补码计算的理解与掌握。其中不乏有很精妙的题目，比如absVal，利用符号位扩展后的0和-1来处理求相反数时按位取反加一的加一操作，使得x与~x+1达成形式上的统一；又比如bitCount，用位运算返回输入x二进制表示中的1的个数，第一次分组计数时选择4个位一组最为合适。如果在选择2个一组，不仅会在不加掩码的情况下有溢出、相互干扰的风险，而且可能遇到最高符号位为1带来的移位困难，而如果选择8个位1组，则会显著提高操作次数，使用更多操作符。

总之本次实验将计算机系统基础课堂上学习的东西用于上机实践，让我掌握了VS2019的编译、链接和调试，各类数据的表示和存储形式，以及地址的计算方法、地址的内存转换，提高了我对位运算的理解，为接下来的理论学习和实验打好了基础。

**五、源码**

实验任务1：compress.c

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<cstring>

using namespace std;

#define N 5

#define N1 2

#define N2 3

struct student {

char name[8];

short age;//2字节，压缩为1字节

float score;//4字节

char remark[200]; // 备注信息

};

/\*以下为输入内容，程序运行后复制以下内容输入即可

tql

19

42

cs2208

hanabi

18

99.6

a

purein

19

86.3

b

npc3

19

64.8

c

npc4

15

59.9

d

\*/

struct student old\_s[N], new\_s[N];

int pack\_student\_bytebybyte(student\* s, int sno, char\* buf)

{//逐字节压缩

int cnt = 0;//计数压缩后字节数

char\* p;//临时指针

// char name[8]

for (int i = 0; i < sno; i++){

p = (char\*)s[i].name;

for (int i = 0; i < 8; i++) {

if (\*p) {

\*(buf++) = \*(p++);//只储存有非0字符，并假设最后一个非零字符后全为0

}

else {

\*(buf++) = 0;//用'\0'或0来标志字符串的结束

break;

}

cnt++;

}

//short age

p = (char\*)&s[i].age;//小端存储，现在指向低位

\*(buf++) = \*(p++);//存低位，舍弃高位

cnt++;

//float score

p = (char\*)&s[i].score;//小段法的表示[DDDDDDDD] [DDDDDDDD] [EDDDDDDD] [SEEEEEEE]

for (int i = 0; i < 4; i++) {

\*(buf++) = \*(p++);

}

cnt += 4;

//char remark[200]

p = (char\*)s[i].remark;

for (int i = 0; i < 200; i++) {

if (\*p) {

\*(buf++) = \*(p++);//只储存有非0字符，并假设最后一个非零字符后全为0

}

else {

\*(buf++) = 0;//用'\0'或0来标志字符串的结束

break;

}

cnt++;

}

}

return cnt;

}

int pack\_student\_whole(student\* s, int sno, char\* buf)

{//一条语句压缩

int cnt = 0; //计数压缩后字节数

char\* p; //临时指针

for (int i = 0; i < sno; i++) {

//name[8]

p = (char\*)s[i].name;

strcpy(buf+cnt, p);

cnt += strlen(p)+1;//strlen()不计入结尾的'/0'

//age

p = (char\*)&s[i].age;

\*(buf + cnt) = \*p;

cnt++;

//score

p = (char\*)&s[i].score;

memcpy(buf + cnt, p, sizeof(float));

cnt += 4;

//remark

p = (char\*)s[i].remark;

strcpy(buf + cnt, p);

cnt += strlen(p) + 1;

}

return cnt;

}

int restore\_student(char\* buf, int len, student\* s)

{//按照成员声明的顺序用一条语句反向解压

int cnt = 0;

int i = 0;

while ( i < len){

strcpy(s[cnt].name, buf + i);

i += strlen(buf + i)+1;

s[cnt].age = \*(buf + i);

i++;

memcpy((char\*)&s[cnt].score, buf + i, sizeof(float));

i += sizeof(float);

strcpy(s[cnt].remark, buf + i);

i += strlen(buf + i) + 1;

cnt++;

}

return cnt;

}

int main()

{

int before\_cnt = 0, after\_cnt = 0;

char message[N \* sizeof(struct student)];

for (int i = 0; i < N; i++){//输入

printf("name:");

scanf("%s", old\_s[i].name);

getchar();

printf("age:");

scanf("%hd", &old\_s[i].age);

getchar();

printf("score:");

scanf("%f", &old\_s[i].score);

getchar();

printf("remark:");

scanf("%s", old\_s[i].remark);

getchar();

}

printf("压缩前的内容如下:\n");

for (int i = 0; i < N; i++){

printf("student %d\nname:%s\nage:%hd\nscore:%f\nremark:%s\n\n",i+1,old\_s[i].name, old\_s[i].age, old\_s[i].score, old\_s[i].remark);

}

before\_cnt = sizeof(old\_s);

after\_cnt = pack\_student\_bytebybyte(&old\_s[0], N1, message);

after\_cnt += pack\_student\_whole(&old\_s[N1], N2, &message[after\_cnt]);

restore\_student(message, after\_cnt, new\_s);

printf("压缩后的内容如下:\n");

for (int i = 0; i < N; i++)

{

printf("student %d\nname:%s\nage:%hd\nscore:%f\nremark:%s\n\n", i + 1, old\_s[i].name, old\_s[i].age, old\_s[i].score, old\_s[i].remark);

}

printf("压缩前存放数据长度:%d\n压缩后存放数据长度:%d\n", before\_cnt, after\_cnt);

printf("输出message的前20个字节内容：");

for (int i = 0; i < 20; i++)

printf("%x", message[i]);

return 0;

}

实验任务2：bits.c

#include<cstdio>

#include<climits>

//(1) int absVal(int x); 返回 x 的绝对值

//仅使用 !、 ~、 & 、^ 、 | 、 + 、 << 、 >> ， 运算次数不超过 10次

int absVal(int x)

{//x -x=~x+1

int mask = x >> 31;//正则全为0，负全为1

return (x ^ mask) + (~mask) + 1;

/\*另一种解法，利用x, x - 2 \* x

int b;

b = (x >> 31) & (x << 1);

return x + (~b) + 1;\*/

}

int absVal\_standard(int x) { return (x < 0) ? -x : x; }

//(2) int negate(int x); 不使用负号，实现 - x

int negate(int x)

{

return ~x + 1;

}

int negate\_standard(int x) { return -x; }

//(3) int bitAnd(int x, int y); 仅使用 ~和 | ，实现&

int bitAnd(int x, int y)

{

return ~(~x | ~y);

}

int bitAnd\_standard(int x, int y) { return x & y; }

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

int bitOr(int x, int y)

{

return ~(~x & ~y);

}

int bitOr\_standard(int x, int y) { return x | y; }

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

int bitXor(int x, int y)

{//x^y = (x&~y)|(~x&y)

return ~(~(x & ~y) & ~(~x & y));

}

int bitXor\_standard(int x, int y) { return x ^ y; }

//(6) int isTmax(int x); 判断x是否为最大的正整数（7FFFFFFF），

//只能使用 !、 ~、 & 、^ 、 | 、 +

int isTmax(int x)

{//只有Tmax满足x+1与~x相等

return !(((x+1)^(~x))|(!(~x)));

}

int isTmax\_standard(int x) { return x==0x7FFFFFFF; }

//(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

int bitCount(int x)

{//注意加括号，+优先级高于&

int a = 0x11 | (0x11 << 8);//00010001 00010001

int mask = a | (a << 16);//00010001 00010001 00010001 00010001

int s;

s = (x & mask) + ((x >> 1) & mask) + ((x >> 2) & mask) + ((x >> 3) & mask);

//此时32位的每连续四位所表示的无符号整数刚好为原x重对应四位中1的个数

// 最大为0100,方便右移

//将8个四位累加起来即可

int b = 0xf | (0xf << 8);//00001111 00001111

int mask2 = b | (b << 16);//00001111 00001111 00001111 0000 1111

s = (s & mask2) + ((s >> 4) & mask2);

//32位的连续8位

//答案最大为32 可用8位二进制数表示，

// 之后不用考虑干扰，直接加

s = s + (s >> 16);

s = s + (s >> 8);

return s&(0xff);

}

int bitCount\_standard(int x) {

int result = 0;

int i;

for (i = 0; i < 32; i++)

result += (x >> i) & 0x1;

return result;

}

//(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。

int bitMask(int highbit, int lowbit)//111100001111

{//构造两个形如11111000类型的数并异或

return (((~0) << highbit)<<1) ^ ((~0) << lowbit);

}

int bitMask\_standard(int highbit, int lowbit)

{

int result = 0;

int i;

for (i = lowbit; i <= highbit; i++)

result |= 1 << i;

return result;

}

//(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

int addOK(int x, int y)

{//溢出时正正得负和负负得正,溢出时返回1

int ans = x + y;

return !!(((x & y & (~ans))| ((~x) & (~y) &ans)) >> 31);

}

int addOK\_standard(int x, int y)

{

long long lsum = (long long)x + y;

return lsum != (int)lsum;

}

//(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

int byteSwap(int x, int n, int m)

{//分别取出除对应字节外其他字节都为0的三部分（第n个字节，第m个字节和其他部分之后移动n与m相加

int max,mask\_n,mask\_m,mask\_nm,ans,sub;

int n8, m8;

//n8 = n << 3;

max = 255;

mask\_n = max<<(n<<3);

mask\_m = max<<(m<<3);

mask\_nm = mask\_n | mask\_m;//形如0000000011111111000011111111000

sub = (m + ~n + 1)<<3;

ans = (x& (~mask\_nm)) | (x & mask\_n)<<sub | ((x & mask\_m)>>sub)&mask\_n ;

return ans;

}

int byteSwap\_standard(int x, int n, int m)

{

/\* little endiamachine \*/

/\* least significant byte stored first \*/

unsigned int nmask, mmask;

switch (n) {

case 0:

nmask = x & 0xFF;

x &= 0xFFFFFF00;

break;

case 1:

nmask = (x & 0xFF00) >> 8;

x &= 0xFFFF00FF;

break;

case 2:

nmask = (x & 0xFF0000) >> 16;

x &= 0xFF00FFFF;

break;

default:

nmask = ((unsigned int)(x & 0xFF000000)) >> 24;

x &= 0x00FFFFFF;

break;

}

switch (m) {

case 0:

mmask = x & 0xFF;

x &= 0xFFFFFF00;

break;

case 1:

mmask = (x & 0xFF00) >> 8;

x &= 0xFFFF00FF;

break;

case 2:

mmask = (x & 0xFF0000) >> 16;

x &= 0xFF00FFFF;

break;

default:

mmask = ((unsigned int)(x & 0xFF000000)) >> 24;

x &= 0x00FFFFFF;

break;

}

nmask <<= 8 \* m;

mmask <<= 8 \* n;

return x | nmask | mmask;

}

void test(int(\*myfuc)(int x), int (\*stdfuc)(int x), int\* data, int num)

{//测试函数，测试1个参数的函数

int ans1, ans2;

for (int i = 0; i < num; i++)

{

printf("测试点:%d ", i + 1);

ans1 = stdfuc(data[i]);

ans2 = myfuc(data[i]);

printf("data:%d 标准输出结果：%d 函数输出结果：%d",data[i], ans1, ans2);

if (ans1 == ans2)

printf(" 输出正确\n");

else

printf(" 输出错误\n");

}

printf("\n");

}

void test(int(\*myfuc)(int ,int), int (\*stdfuc)(int ,int), int data[][2], int num)//2 parameters

{//测试函数，测试2个参数的函数

int ans1, ans2;

for (int i = 0; i < num; i++)

{

printf("测试点:%d ", i + 1);

ans1 = stdfuc(data[i][0],data[i][1]);

ans2 = myfuc(data[i][0],data[i][1]);

printf("data:%d %d 标准输出结果：%d 函数输出结果：%d", data[i][0],data[i][1], ans1, ans2);

if (ans1 == ans2)

printf(" 输出正确\n");

else

printf(" 输出错误\n");

}

printf("\n");

}

void test(int(\*myfuc)(int, int,int), int (\*stdfuc)(int, int,int), int data[][3], int num)//2 parameters

{//测试函数，测试3个参数的函数

int ans1, ans2;

for (int i = 0; i < num; i++)

{

printf("测试点:%d ", i + 1);

ans1 = stdfuc(data[i][0], data[i][1],data[i][2]);

ans2 = myfuc(data[i][0], data[i][1],data[i][2]);

printf("data:0x%x %d %d 标准输出结果：0x%x 函数输出结果：0x%x", data[i][0], data[i][1], data[i][2],ans1, ans2);

if (ans1 == ans2)

printf(" 输出正确\n");

else

printf(" 输出错误\n");

}

printf("\n");

}

int main()

{

printf("开始测试\n");

int ans1, ans2;//以下为测试数据，已尽量考虑特殊值,极端情况

int test1[4] = { 5,-9,0,INT\_MIN };

int test2[4] = { 5,-9,0,INT\_MIN };

int test3[3][2] = { {842616,55231},{ -135469,-151561 },{0,31249} };

int test4[3][2] = { {842616,55231},{ -135469,-151561 },{0,31249} };

int test5[3][2] = { {842616,55231},{ -135469,-151561 },{0,31249} };

int test6[3] = { 59531,INT\_MAX,-1 };

int test7[3] = { 0,INT\_MIN,-1 };

int test8[3][2] = { {3,0},{ 26,5 },{31,9} };

int test9[3][2] = { {6551,-5651},{ 65,INT\_MAX },{-9,INT\_MIN} };

int test10[3][3] = { {0xABCDEDCB,1,3},{ 0xABCDEDCB,1,1 },{0xABCDEDCB,0,2} };

char str[11][100] = { " ","absVal", "negate", "bitAnd", "bitOr", "bitXor", "isTmax", "bitCount", "bitMask", "addOK", "byteSwap"};

for (int i = 1; i <= 10; i++)

{

printf("测试函数%d: %s\n", i,str[i]);

switch (i)

{

case 1:

test(absVal, absVal\_standard, test1, 4);

break;

case 2:

test(negate, negate\_standard, test2, 4);

break;

case 3:

test(bitAnd, bitAnd\_standard, test3, 3);

break;

case 4:

test(bitOr, bitOr\_standard, test4, 3);

break;

case 5:

test(bitXor, bitXor\_standard, test5, 3);

break;

case 6:

test(isTmax, isTmax\_standard, test6, 3);

break;

case 7:

test(bitCount, bitCount\_standard, test7, 3);

break;

case 8:

test(bitMask, bitMask\_standard, test8, 3);

break;

case 9:

test(addOK, addOK\_standard, test9, 3);

break;

case 10:

test(byteSwap, byteSwap\_standard, test10, 3);

break;

}

}

printf("测试结束\n");

return 0;

}