

**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 ： 计算机2305班**

**学 号 ： U202315677**

**姓 名 ： 岳皓**

**指导教师 ： 班鹏新**

**2025 年 3 月 14 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=4)

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==2）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30个字节的内容，并与调试时在内存窗口观察到的message的前30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 bitNor(int x, int y) 仅使用 ~ 和 &，实现或非↓
2. int isTmin(int x) 判断x是否为最小的负整数（0x80000000），

只能使用 !、 ~、 &、 ^、 |、 +

1. int isEqual(int x, int y) 当x与y相等时返回1，否则返回0

只能使用!、 ~、 &、 ^、 |、 +、 <<、 >>，运算次数不超过5次

1. int logicalShift(int x, int n) x逻辑右移n（0<=n<=31）位。只能使用! ~ & ^ | + << >>，运算次数不超过 20次。例如logicalShift(0x87654321,4) = 0x08765432
2. 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。
3. int subOK(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**

逐个压缩函数pack\_student\_bytebybyte的思想大致如下：

将一个结构体数组（student）序列化到一个字节缓冲区(buf）中。将每个数据按顺序一个一个字节写入缓冲区中。

该函数接受三个参数，分别为：student\* s（指向学生结构体数组的指针）、sno（数组中学生的数量）、char\* buf（用于存储序列化数据的缓冲区）。

初始化：cnts=0（用于跟踪记录已处理的学生数），cntname、cntage、cntscore、cntremark均初始化为0，分别用于记录压缩循环过程中结构体数组中name, age, score, remark字段中数据的字节数以便于跳过剩余部分。cntbuf=0用于记录累计写入缓冲区的总字节数，以及两个指针p=（char\*）s（指向学生数组的起始地址）、p0=buf（指向缓冲区的起始地址）。

基本过程：通过循环依次处理每个学生，首先通过while(cnts<sno)循环遍历每个学生，对于name和remark字段，均使用while循环读取每个字节直到遇到字符串终止符\0，然后填充0的同时调整指针跳过剩余字节。对于age和score字段，则分别直接循环读取2个和4个字节。函数最后返回cntbuf，写入缓冲区的总字节数。

整体压缩函数pack\_student\_whole的思想大致如下：

将一个结构体数组（student）序列化到一个字符缓冲区（buf）中，遍历数组并将每个数据按顺序利用strcpy函数复制到提供的缓冲区中。同逐个压缩函数一样接受三个参数：student\* s（指向学生结构体数组的指针）、sno（数组中学生的数量）、char\* buf（用于存储序列化数据的缓冲区）。

初始化：i=0（用于遍历学生数组的索引）、char\* p = NULL（临时指针，用于指向当前处理的字符串或数据）、char\* p0 = buf（指向缓冲区的当前写入位置的指针，初始时指向缓冲区的开始）。

基本过程：通过循环依次处理每个学生数据，首先通过while(i<sno)遍历所有学生，对于name和remark字段，使用函数strcpy复制到缓冲区当前位置，再更新p0指针，使其指向字符串的下一个位置，同时考虑到字符串末尾的空字符，应该使p0移动strlen(p)+1位。对于age和score字段，将数据直接复制到缓冲区的当前位置，再向后移动相应的字节。函数最后返回p0-buf，即序列化数据在缓冲区中占据的总字节数。

解压函数restore\_student的思想大致如下：

从一个字符缓冲区buf中解压出学生数据，并将这些数据填充到一个student类型的数组s中。缓冲区中的数据是按照特定的格式存储的，即每个学生的数据依次包括名字（字符串）、年龄（短整型）、分数（浮点型）和备注（字符串）。函数会遍历缓冲区，逐个解析出每个学生的数据，并填充到数组s中。函数最后返回解压出的学生数量。

该函数接受三个参数：char\* buf（指向存储压缩数据的缓冲区的指针）。int len（缓冲区中数据的长度）。student\* s（指向用于存储解压数据的student类型数组的指针)。

初始化：int i = 0（用于记录解压出的学生数量，也作为数组s的索引）。char\* p = buf（指向缓冲区当前位置的指针，初始时指向缓冲区的开始）。

基本过程：通过while（p-buf<len）遍历数据，对于name和remark字段，使用函数strcpy

将缓冲区当前位置的字符串分别复制到s[i].name和s[i].remark中，再更新指针p,使其指向字符串的下一个位置，同时考虑到字符串末尾的空字符，应让p分别移动strlen(s[i].name)+1和strlen(s[i].remark)+1位。对于age和score字段，分别将数据转化为短整型和浮点型后分别复制到s[i].age和s[i].score中。然后使i加一，进入下一次循环。函数最后返回i，即解压出的学生数量。

对程序进行测试，输入数据如图1，输出结果如图2和图3。

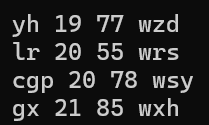
****

图 1 数据输入

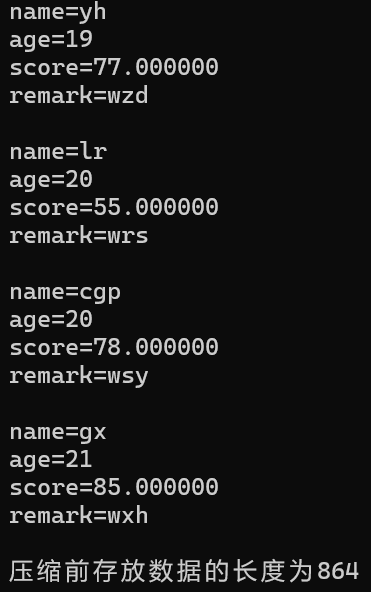
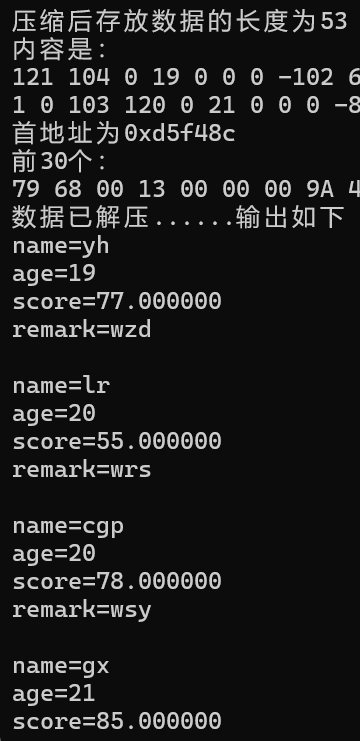
 

图 2 压缩前输出 图 3 压缩后输出

其中以十六进制的形式，输出message的前30个字节的内容以及其在内存窗口的显示结果如图4，图中输出message的首地址0x98eff290，再输出前30个数据。

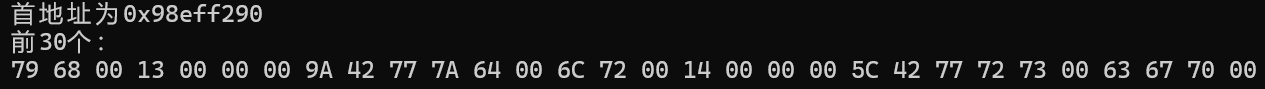


图 4 message前30个字节的输出

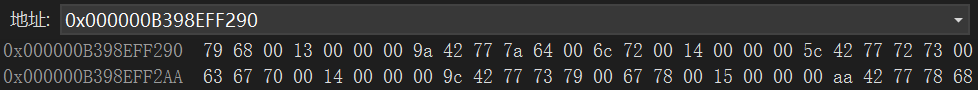


图 5 message在内存中的显示

通过比较message输出和内存中相对应的前30个数据，发现两者结果相一致，输出结果正确。

对于第0个学生的score，其值为77，转换成二进制表示为1001101，进行规格化为1.001101，移动6位，故符号位为0，阶码为127+6=133即10000101，尾数为0011 0100 0000 0000 0000 000，转为十六进制为0x429A0000，由于采用小端存储，故内存内应为00 00 9A 42，这与内存显示中的score部分相一致，如图6。

QQ20250313-193116

图 6 第0个同学的内容在内存的显示

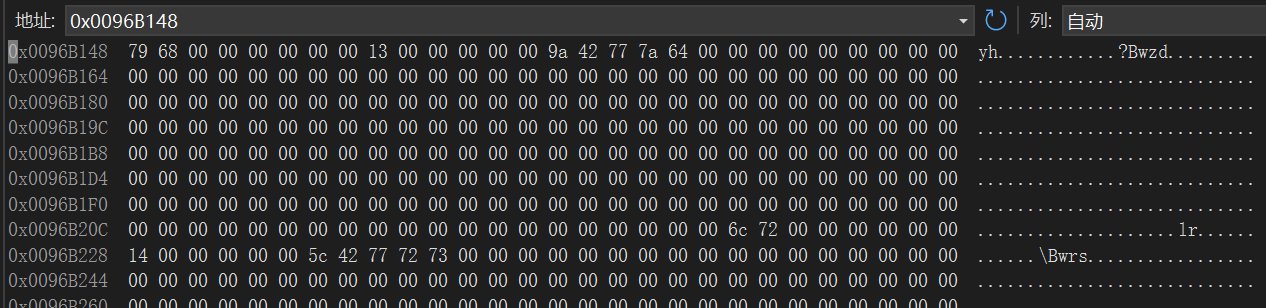


图 7 结构数组的存储显示

观察图7中结构数组在内存中的存储情况可知，每个数据字段都是相邻存放的，前8个字节为char类型的name[8]，由于char占一个字节，故其内容按顺序存放，后面四个字节在内存中用于存放short类型的数，采用小端存储，再后面四个字节为float类型的数，采用小端存储，然后，后面200个字节为char类型的remark[200]，按顺序存放。

**（2）任务 2 的算法思想、运行结果等记录**

各函数极其思想如下：

1.absVal(int x):利用符号位扩展，将x右移31位后赋值给mask，如果x为负数，则mask为全1，否则为0，然后返回（x+mask)^mask，如果x为负数，则x+mask相当于x-1，然后与mask(=1)异或，相当于取反，如果x为非负数，mask为0，返回x。

2.negate(int x)：利用补码性质，直接返回~x+1，即x取反加一。

3.bitAnd(int x,int y):利用德摩根定律x&y=~(~x|~y),直接返回~（~x|~y）。

4.bitNor(int x,int y):或非操作使将两者进行|操作然后结果取反，所以直接返回~（x|y）。

5.isTmin(int x):int类型的最小数是0x80000000，其特点是与自己相加为0，所以直接判断x+x是否为0，同时还要判断x本身是否为0，若前者为0，后者不为0，则说明x为最小值，否则不是最小值。

6.isEqual(int x,int y):利用异或操作，若x与y相等，则x^y为0，若不相等则为1，对结果取反就是所求，所以直接返回！(x^y)。

7.logicalShift(int x,int y):逻辑右移需要在高位补0而非符号位，故初始化一个int类型变量mask，将1左移31位得到0x80000000，再右移n位并左移1位后取反以得到高32-n位为0且低n位为0的值并赋给mask，将x算术右移后再与mask相与来清除高位的符号位从而得到结果。

8.bitMask(int highbit,int lowbit):分别构造从第0位到highbit位位1的值和从lowbit到最高位位1的值，将两者相与则得到结果。

9.subOK(int x,int y):首先计算x-y的值，由于不能使用-，所以将y取反加一，然后分别对x,y以及x-y向右移31位，得到三者的符号位，比较x与y的符号位是否不同，且结果的符号位与x的符号位是否不同，如均不相同，则发生溢出。

10.byteSwap(int x,int n,int m):分别提取x的第n个字节和第m个字节，分别将n和m扩大8倍，再分别将x右移相应扩大后的数并与0xFF相与，再分别将x对应的字节清零后替换，其中，byteN记录第n个字节,byteM记录第m个字节，分别左移相应的位数与相应位数清零的x相或即可。

测试结果：

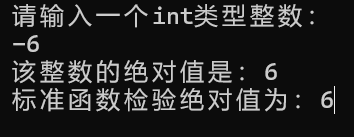
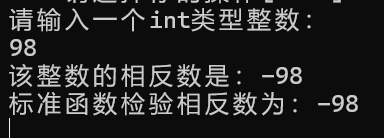
 

图 8 函数运行结果 图 9 函数2运行结果

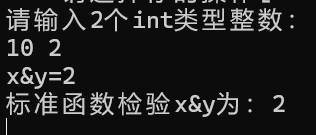
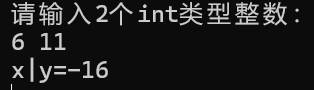
 

图 10 函数3运行结果 图 11 函数4运行结果

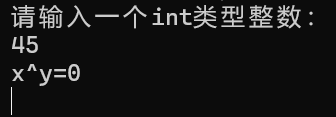
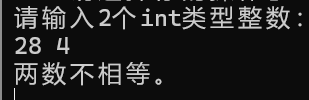
 

图 12 函数5运行结果 图 13 函数6运行结果

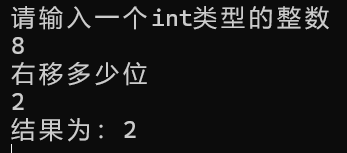
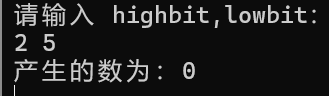
 

图 14 函数7运行结果 图 15 函数8运行结果1

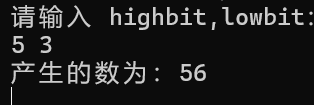
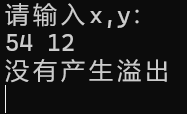
 

图 16 函数8运行结果2 图 17 函数9运行结果

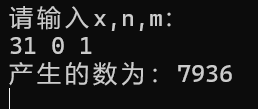


图 18 函数10运行结果

**四、体会**

通过实验一的两次任务，我更加深入地了解了数据在计算机系统中的存储方式和表示形式，并通过对代码的系列操作和改进，我更进一步了解vs的使用，学会了调试程序以及对于内存的查看和分析，为以后的学习夯实基础，同时更加熟悉地址的操作以及体会到vs的便捷，不仅帮助我快速定位并修复一系列bug，还让我对代码的运行逻辑有了更加深入的了解，但与此同时，对于该集成开发环境的使用还需要进一步学习和实操，以便更好利用。

此外，我还深入理解了整型、浮点型、字符型等数据在内存中的表示方式以及存储方式，然我体会到学以致用的快乐，同时，也让我重温了一遍位运算的各种知识，让我更加熟练的使用位运算来编写更加高效的程序。

**五、源码**

实验任务 1、2 的源程序

任务1：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<bits/stdc++.h>

using namespace std;

#define N 4

#define N1 2

#define N2 2

struct student {

char name[8];

short age;

float score;

char remark[200];

}old\_s[N],new\_s[N];

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

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

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

int main() {

int cnts = 0;

char message[600];

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

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

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

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

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

}

// 输出压缩前的结果

while (cnts < N) {

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++;

}

cnts = 0;

printf("压缩前存放数据的长度为%lu\n", sizeof(struct student) \* N);

int cntbuf = pack\_student\_bytebybyte(old\_s, N1, message);

cntbuf += pack\_student\_whole(old\_s + 2, N2, message + cntbuf);

printf("\n压缩后存放数据的长度为%d\n内容是：\n", cntbuf);

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

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

}

printf("\n首地址为0x%x\n前30个：\n", &message);

for (int i = 0; i < 30; i++) {

printf("%02X ", (unsigned char)message[i]);

}

int sno = restore\_student(message, cntbuf, new\_s);

printf("\n数据已解压......输出如下\n");

cnts = 0;

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");

cnts++;

}

return 0;

}

int pack\_student\_bytebybyte(struct 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) {

cntbuf++;

if (\*p) {

\*p0 = \*p;

cntname++;

p++;

p0++;

}

else {

\*p0 = 0;

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) {

cntbuf++;

if (\*p) {

\*p0 = \*p;

cntremark++;

p++;

p0++;

}

else {

\*p0 = 0;

p += (200 - cntremark);

p0++;

break;

}

}

cnts++;

}

return cntbuf;

}

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

int i = 0;

char\* p = NULL;

char\* p0 = buf;

while (i < sno) {

p = s[i].name;

strcpy(p0, p);

p0 += (strlen(p) + 1); // 压缩name

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

\*((short\*)p0) = \*((short\*)p);

p0 += 2; // 压缩age

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

\*((float\*)p0) = \*((float\*)p);

p0 += 4; // 压缩score

p = s[i].remark;

strcpy(p0, p);

p0 += (strlen(p) + 1); // 压缩remark数组

i++;

}

return p0 - buf;

}

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

int i = 0;

char\* p = buf;

while (p - buf < len) {

strcpy(s[i].name, p); // 读入name

p += strlen(s[i].name) + 1;

s[i].age = \*((short\*)p); // 读入age

p += 2;

s[i].score = \*((float\*)p); // 读入score

p += 4;

strcpy(s[i].remark, p); // 读入remark

p += strlen(s[i].remark) + 1;

i++;

}

return i; // 返回解压人数

}

任务2：

#include<bits/stdc++.h>

using namespace std;

// 1

int absVal(int x) {

int mask = x >> 31; // 如果x为负，mask为全1；否则为全0

return (x + mask) ^ mask; // 如果x为负，取反加1；否则不变

}

// 判断函数

int absVal\_standard(int x) {

return (x < 0) ? -x : x;

}

// 2

int lnegate(int x) {

return ~x + 1; // 取反加1即为负数

}

// 判断函数

int lnegate\_standard(int x) {

return -x;

}

// 3

int bitAnd(int x, int y) {

return ~(~x | ~y); // 德摩根定律：x & y = ~(~x | ~y)

}

// 判断函数

int bitAnd\_standard(int x, int y) {

return x & y;

}

// 4

int bitNor(int x, int y) {

return ~(x | y); // 或非操作

}

// 5

int isTmin(int x) {

return !(x + x) & !!(x); // Tmin是0x80000000，唯一满足x + x == 0且x != 0

}

// 6

int isEqual(int x, int y) {

return !(x ^ y); // 如果x == y，x ^ y为0，取反为1

}

// 7

int logicalShift(int x, int n) {

int mask = ~(((1 << 31) >> n) << 1); // 构造掩码，保留高32-n位

return (x >> n) & mask;

}

// 8

int bitMask(int highbit, int lowbit) {

return ((1 << (highbit + 1)) - 1) & ~((1 << lowbit) - 1); // 构造掩码

}

// 9

int subOK(int x, int y) {

int diff = x + (~y + 1); // x - y

int signX = x >> 31; // x的符号位

int signY = y >> 31; // y的符号位

int signDiff = diff >> 31; // diff的符号位

int overflow = (signX ^ signY) & (signX ^ signDiff);

return !!overflow; // 将结果转换为0或1

}

// 10

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

int byteN = (x >> (n << 3)) & 0xFF; // 获取第n个字节

int byteM = (x >> (m << 3)) & 0xFF; // 获取第m个字节

int maskN = 0xFF << (n << 3); // 第n个字节的掩码

int maskM = 0xFF << (m << 3); // 第m个字节的掩码

x = (x & ~maskN) | (byteM << (n << 3)); // 替换第n个字节

x = (x & ~maskM) | (byteN << (m << 3)); // 替换第m个字节

return x;

}

int main() {

int op = 1;

while (op) {

system("cls"); printf("\n\n");

printf(" Menu for Bit Calculation \n");

printf("-------------------------------------------------\n");

printf(" 1. absVal 6. isEqualv\n");

printf(" 2. negate 7. logicalShift\n");

printf(" 3. bitAnd 8. bitMask\n");

printf(" 4. bitNor 9. subOK\n");

printf(" 5. isTmin 10. byteSwap\n");

printf(" 0. Exit\n");

printf("-------------------------------------------------\n");

printf(" 请选择你的操作[0~10]:");

cin >> op;

switch (op) {

case 1: {

int x;

cout << "请输入一个int类型整数：" << endl;

cin >> x;

cout << "该整数的绝对值是：" << absVal(x) << endl;

cout << "标准函数检验绝对值为：" << absVal\_standard(x);

getchar(); getchar();

break;

}

case 2: {

int x;

cout << "请输入一个int类型整数：" << endl;

cin >> x;

cout << "该整数的相反数是：" << lnegate(x) << endl;

cout << "标准函数检验相反数为：" << lnegate\_standard(x) << endl;

getchar(); getchar();

break;

}

case 3: {

int x, y;

cout << "请输入2个int类型整数：" << endl;

cin >> x >> y;

cout << "x&y=" << bitAnd(x, y) << endl;

cout << "标准函数检验x&y为：" << bitAnd\_standard(x, y) << endl;

getchar(); getchar();

break;

}

case 4: {

int x, y;

cout << "请输入2个int类型整数：" << endl;

cin >> x >> y;

cout << "x|y=" << bitNor(x, y) << endl;

getchar(); getchar();

break;

}

case 5: {

int x;

cout << "请输入一个int类型整数：" << endl;

cin >> x;

cout << "x^y=" << isTmin(x) << endl;

getchar(); getchar();

break;

}

case 6: {

int x, y;

cout << "请输入2个int类型整数：" << endl;

cin >> x >> y;

if (isEqual(x,y))cout << "两数相等。" << endl;

else cout << "两数不相等。" << endl;

getchar(); getchar();

break;

}

case 7: {

int x,n;

cout << "请输入一个int类型的整数" << endl;

cin >> x;

cout << "右移多少位" << endl;

cin >> n;

cout << "结果为：" << logicalShift(x,n) << endl;

getchar(); getchar();

break;

}

case 8: {

int x, y;

cout << "请输入 highbit,lowbit：" << endl;

cin >> x >> y;

cout << "产生的数为：" << bitMask(x, y) << endl;

getchar(); getchar();

break;

}

case 9: {

int x, y;

cout << "请输入x,y：" << endl;

cin >> x >> y;

if (subOK(x, y))cout << "产生了溢出" << endl;

else cout << "没有产生溢出" << endl;

getchar(); getchar();

break;

}

case 10: {

int x, m, n;

cout << "请输入x,n,m：" << endl;

cin >> x >> n >> m;

cout << "产生的数为：" << byteSwap(x, n, m) << endl;

getchar(); getchar();

break;

}

}

}

return 0;

}