

**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 ： CS2204**

**学 号 ： U202215458**

**姓 名 ： 柯晓豪**

**指导教师 ： 金良海**

**2024 年 5 月 29 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[5], new\_s[5];

struct student {

char name[8];

short age;

float score;

char remark[200]; // 备注信息

};

编写程序，输入5个学生的信息到结构数组old\_s中。将 old\_s[5] 中的所有信息依次紧凑存放到一个字符数组message中，然后从 message 转换到结构数组 new\_s[5]中。打印压缩前、解压后的结果，以及压缩前、压缩后存放数据的长度。

要求：

1. 输入的第0个人姓名为自己的名字；
2. 要求使用指定的函数完成数据压缩

压缩函数的格式为： 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 为学生人数； buf 为压缩存储区的首地址；返回压缩后的字节数。

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。前2个记录压缩调用pack\_student\_bytebybyte 完成；后3个记录压缩调用pack\_student\_whole。两种压缩函数都只调用1次。

（3）要求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。

要求：观察压缩前结构数组在内存中的存放结果；指出结构中各个字段的相对位置关系；压缩后的数据在内存中的存放结果。  
 （4）仿照调试时没看到的内存数据，以十六进制的形式，输出message前的20个字节的内容，并与调试时在内存窗口观察到的mesaage的前20个字节比较是否一致。

**任务2 编写数据表示的自动评测程序**

① 程序随机产生一个整数（可正、可负、为0），在屏幕上显示出来，由用户输入其补码表示（int ， 4个字节，以十六进制形式输入），由程序给出用户输入结果是否正确的判断；

② 程序随机产生有符号整型（4字节）数的补码，由用户输入其对应的十进制数；由程序给出用户输入结果是否正确的判断。

任务3 编写程序，用指定的操作完成给定的功能，并自动判断程序的运行结果是否正确。

1. int negate(int x); 不使用负号，实现 -x
2. int bitAnd(int x, int y); 仅使用 ~ 和 |，实现 &
3. int bitXor(int x, int y); 仅使用 ~ 和 &，实现 ^
4. 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。

**三、实验记录及问题回答**

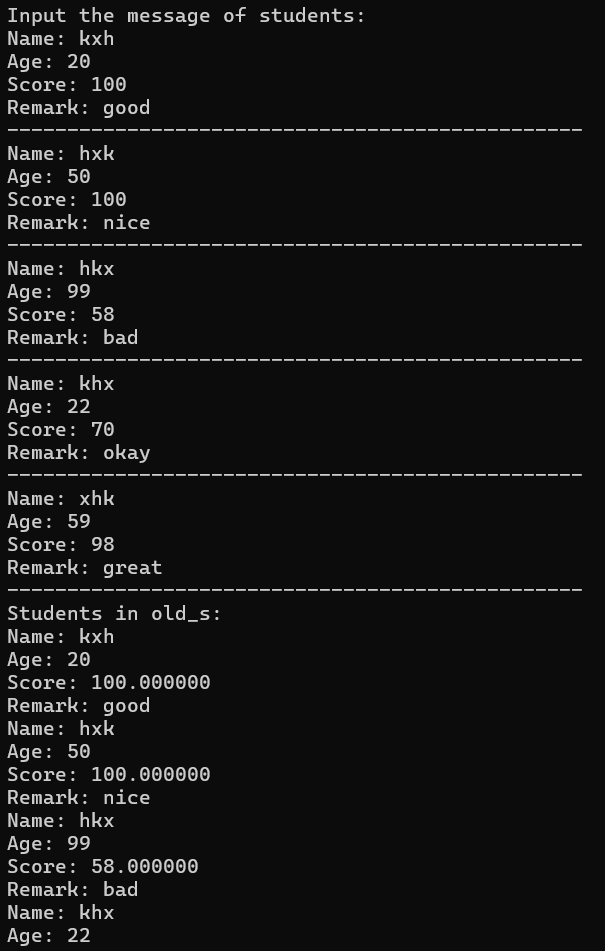
**（1）任务 1 的算法思想、运行结果等记录**

任务1的算法思想如下：

pack\_student\_bytebybyte:该函数使用一个char型指针遍历结构体中的数据。第一个数据成员是char\*类型的变量name，其大小为8个字节，使用for循环遍历每个字节，当遍历到’\0’时，意味着已读入name，此时将指针移动到第二个数据成员age。注意，结构体的数据存储方式可能存在数据对齐，通过查阅资料得知可以使用offsetof方法定位到下一个数据成员的起始位置。其他数据成员的压缩过程类似，也是使用前面定义好的char型指针遍历数据。当一个结构体变量遍历完成后，继续下一个结构体变量的压缩。

pack\_student\_whole:该函数同样使用一个char型指针，只不过这次在压缩数据时是以整个数据变量进行压缩，而不是像pack\_student\_bytebybyte一样，一个一个字节压缩。通过将指针进行强制类型转换实现。字符串变量name比较特殊，可以直接调用strcpy实现。同样地，我们需要调用offsetof方法定义到下一个数据成员的起始位置。

restore\_student:该函数采用与压缩函数相反的方法进行解压。读取buf中的数据，并赋值给结构体变量，直到buf读完为止。运行结果如图1。可以看到，在经过压缩和解压后，数据内容不变，任务基本完成。

****

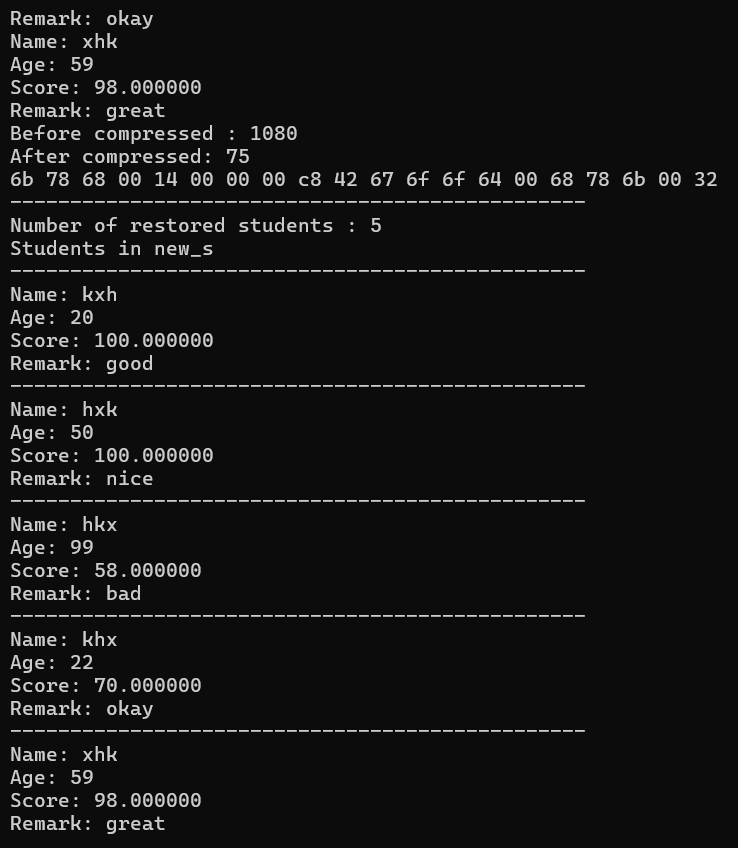
****

图1 任务1运行结果截图

**（2）任务 2 的算法思想、运行结果等记录**

absVal：首先，通过位移操作 x >> 31赋值给p，结果为0则为正数，所有位上全是1则为负数。取反 ~p并对x进行按位与操作(~p & x)，这一步的目的是使得x为正数时保留x。p 与 x 进行按位与操作p & ~x，这一步的目的是使得x为负数时取得 ~x 。将上述两步得到的结果相加 (~p & x) | (p & ~x)，如果 x 是负数，还需要加1，通过 !!p 可以巧妙地实现这一点。

negate：取反加1即可。

bitAnd:利用德摩根定律可得。

bitOr:利用德摩根定律可得。

bitXor: 首先，通过 x & ~y 得到了 x 中存在而 y 中不存在的位。然后，通过 ~x & y 得到了 y 中存在而 x 中不存在的位。最后，将这两者合并起来，即 (x & ~y) | (~x & y)，得到了两个整数按位异或的结果。

isTmax: 首先，通过 (x + 1) 将 x 的值加一。由于最大的正整数加一会导致溢出成为负数，所以 (x + 1) 的结果会是一个负数。对 (x + 1) 进行按位取反 ~(x + 1)，得到的是一个特定的数值，如果 x 是最大的正整数，那么这个数值就是 0x80000000。进行异或操作 ~(x + 1) ^ x，如果 x 是最大的正整数，那么异或的结果将全为 1。再通过 !!(x + 1) 判断 (x + 1) 是否为零。如果 x 是最大的正整数，那么 (x + 1) 为零，取反后为真。

最后，通过逻辑与操作 !(~(x + 1) ^ x) & !!(x + 1)，当 (x + 1) 不为零且 (x + 1) 与 x 的异或结果全为 1 时，返回真。由于本人之前在课外学习时做过这道题，因为当时那道题是不允许直接使用0x7fffffff这种大整数的，于是便采用了这种相对复杂的方法。

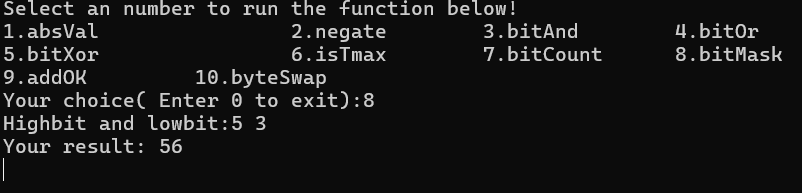
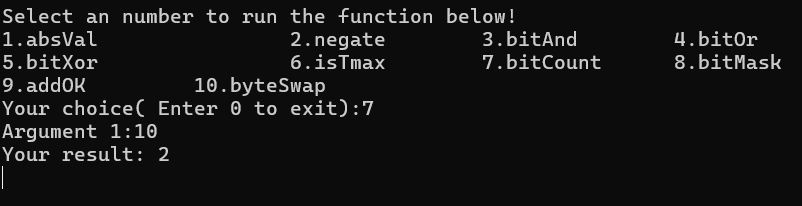
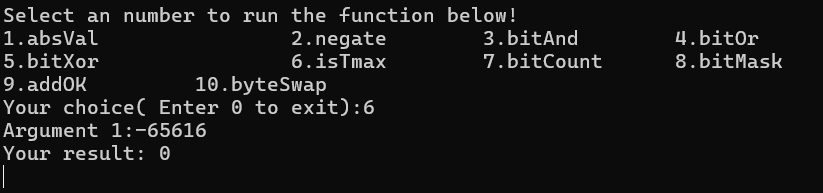
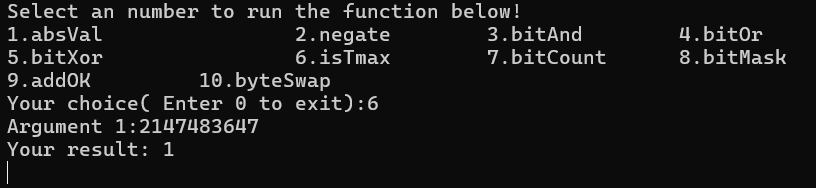
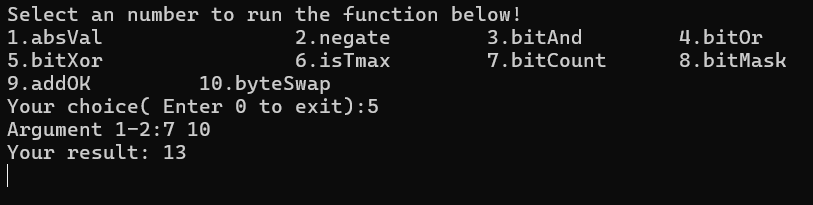
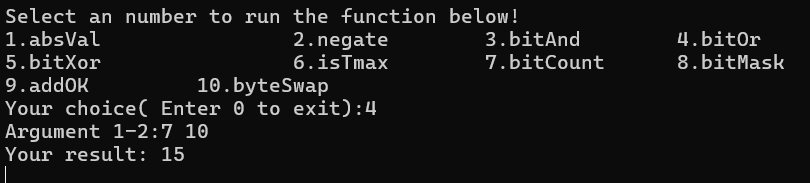
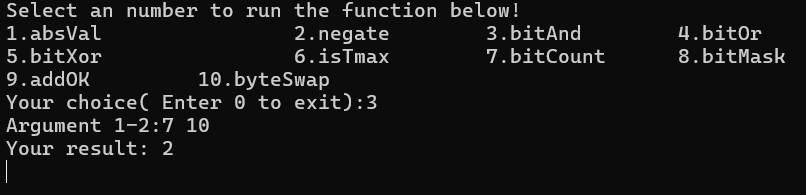
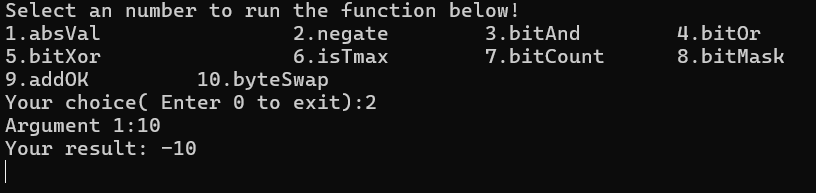
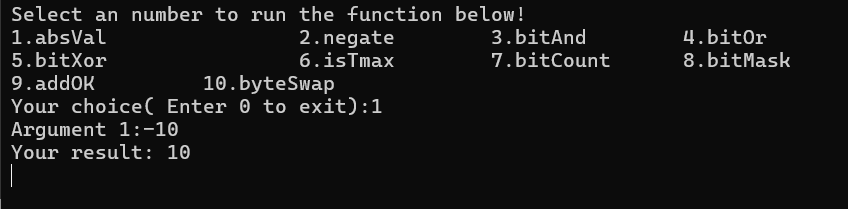
bitCount: 首先，通过 (x >> 1) & 0x55555555 操作，将 x 中相邻的 2 位中的 1 的个数相加，并将结果存储到 x 中。接着，通过 (x & 0x33333333) + ((x >> 2) & 0x33333333) 操作，将 x 中相邻的 4 位中的 1 的个数相加，并将结果存储到 x 中。然后，通过 (x + (x >> 4)) & 0x0f0f0f0f 操作，将 x 中相邻的 8 位中的 1 的个数相加，并将结果存储到 x 中。进行两次右移操作 (x >> 8) 和 (x >> 16)，分别将 x 中相邻的 16 位和 32 位中的 1 的个数相加，并将结果存储到 x 中。最后，通过 x & 0x3f 操作，取 x 中的低 6 位，即得到了整数 x 的二进制表示中 1 的个数。

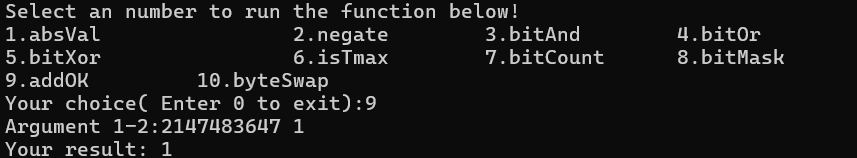
bitMask: 首先，通过 ~0 << lowbit 得到一个所有位都为 1，但是低位到 lowbit（含）之间的位都为 0 的数，这是通过左移操作得到的。然后，通过 ~0 << (highbit + 1) 得到一个所有位都为 1，但是低位到 (highbit + 1)（含）之间的位都为 0 的数，这同样是通过左移操作得到的。对这两个数分别取反，得到低位到 lowbit（含）和 (highbit + 1)（含）之间的位都为 0，其他位都为 1 的数。最后，将这两个数进行按位与操作 l & h，即可得到从 lowbit 到 highbit 全为 1，其他位为 0 的数。

addOK: 首先，将 x 和 y 相加得到结果 res。分别获取 x、y、res 的符号位，通过右移操作将符号位移到最低位。通过异或操作 x\_s ^ y\_s 得到 x 和 y 的符号是否相同的标志，通过异或操作 x\_s ^ res\_s 得到 x 和 res 的符号是否相同的标志。通过逻辑非操作 !! 将标志转换为 0 或 1。判断标志 tag1 和 tag2，如果 tag1 为 1，表示 x 和 y 的符号不同，如果 tag2 为 0，表示 x 和 res 的符号不同。如果 tag1 和 tag2 同时满足条件，则说明会产生溢出，返回 1；否则返回 0。最后，通过取反操作 !(tag1 | tag2) 实现返回结果。

byteSwap: 首先，通过 (x >> (n << 3)) & 0xff 获取整数 x 中第 n 个字节的值，并存储到变量 a 中。然后，通过 (x >> (m << 3)) & 0xff 获取整数 x 中第 m 个字节的值，并存储到变量 b 中。将整数 x 中第 n 和第 m 个字节的位置清零，通过 ~(0xff << (n << 3)) 和 ~(0xff << (m << 3)) 得到掩码，然后通过按位与操作将这两个字节的位置清零。将变量 a 左移 m << 3 位，并将结果按位或到 x 上，即将第 n 个字节的值移动到了第 m 个字节的位置。将变量 b 左移 n << 3 位，并将结果按位或到 x 上，即将第 m 个字节的值移动到了第 n 个字节的位置。最后返回交换后的结果。

运行结果如图2所示。





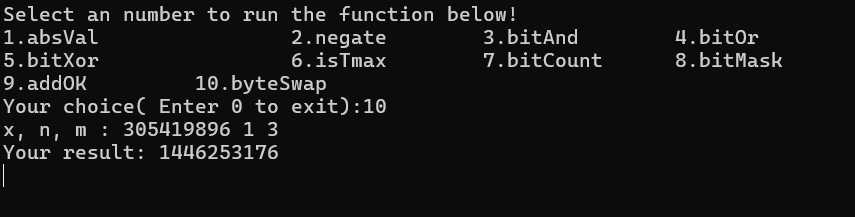
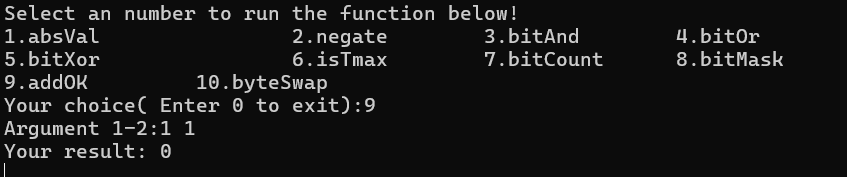


图2 任务2运行截图

注：305419896是0x12345678的十进制数，1446253176是0x56341278的十进制数。

**四、体会**

通过本次实验，我对数据在计算机中的表示以及存储有了一定的了解，对结构体数据成员的存储对齐有了较深的理解，明白了计算机对一些基本的运算的位运算实现，并能通过C语言编写位运算代码。

**五、程序清单**

**任务1：**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stddef.h>

#define N 5//5

#define N1 2//2

#define N2 3//3

struct student {

char name[8];

short age;

float score;

char remark[200]; // 备注信息

};

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);

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

char\* message;

int main() {

int len\_original = sizeof(old\_s);

printf("Input the message of students:\n");

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

printf("Name: ");

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

printf("Age: ");

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

printf("Score: ");

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

printf("Remark: ");

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

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

}

printf("Students in old\_s:\n");

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

{

printf("Name: %s\n", old\_s[i].name);

printf("Age: %d\n", old\_s[i].age);

printf("Score: %f\n", old\_s[i].score);

printf("Remark: %s\n", old\_s[i].remark);

}

printf("Before compressed : %d\n", len\_original);

message = (char\*)malloc(len\_original);

int len1 = pack\_student\_bytebybyte(old\_s, N1, message);

int len2 = pack\_student\_whole(&old\_s[N1], N2, message + len1);

int len\_after = len1 + len2;

printf("After compressed: %d\n", len\_after);

int num = restore\_student(message, len\_after, new\_s);

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

printf("%02x ", message[i]&0xFF);

}

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

printf("Number of restored students : %d\n", num);

printf("Students in new\_s\n");

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

{

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

printf("Name: %s\n", new\_s[i].name);

printf("Age: %d\n", new\_s[i].age);

printf("Score: %f\n", new\_s[i].score);

printf("Remark: %s\n", new\_s[i].remark);

}

return 0;

}

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

int count = 0, index;

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

{

index = 0;

struct student e = \*s;

char\* arr = (char\*)&e;

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

{

if (arr[index] == '\0') {

index++;

continue;

}

\*buf = arr[index];

buf++, index++;

count++;

}

\*buf = '\0';

buf++;

count++;

index = offsetof(struct student, age);

for (int i = 0; i < 2; i++)

{

\*buf = arr[index];

buf++, index++;

}

index = offsetof(struct student, score);

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

{

\*buf = arr[index];

buf++, index++;

}

count += 6;

index = offsetof(struct student, remark);

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

{

if (arr[index] == '\0') {

break;

}

\*buf = arr[index];

buf++, index++;

count++;

}

\*buf = '\0';

count++, buf++;

s++;

}

return count;

}

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

{

int count = 0, index;

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

{

index = 0;

struct student e = \*s;

char\* arr = (char\*)&e;

strcpy(buf, arr);

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

count += (strlen(arr) + 1);

index = offsetof(struct student, age);

\*(short\*)buf = \*(short\*)&arr[index];

buf += 2;

index = offsetof(struct student, score);

\*(float\*)buf = \*(float\*)&arr[index];

buf += 4;

count += 6;

index = offsetof(struct student, remark);

strcpy(buf, &arr[index]);

buf += (strlen(&arr[index]) + 1);

count += (strlen(&arr[index]) + 1);

s++;

}

return count;

}

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

{

int num = 0, count = 0;

while (count < len)

{

num++;

char name[8];

strcpy(name, buf);

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

count += (strlen(name) + 1);

short age = \*(short\*)buf;

buf += 2;

float score = \*(float\*)buf;

buf += 4;

count += 6;

char remark[200];

strcpy(remark, buf);

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

count += (strlen(remark) + 1);

s->age = age;

strcpy(s->name, name);

s->score = score;

strcpy(s->remark, remark);

s++;

}

return num;

}

**任务2：**

#include <iostream>

#include <algorithm>

#include <Windows.h>

using namespace std;

/\*

\* (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;}

\*/

int absVal(int x) {

int p = x >> 31;

return (~p & x) | (p & ~x) + !!p;

}

int absVal\_standard(int x) {

return abs(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;}

\*/

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 &

判断函数： int bitAnd\_standard(int x, int y) { return x & y;}

\*/

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) {

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) {

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

}

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) {

x = x - ((x >> 1) & 0x55555555);

x = (x & 0x33333333) + ((x >> 2) & 0x33333333);

x = (x + (x >> 4)) & 0x0f0f0f0f;

x = x + (x >> 8);

x = x + (x >> 16);

return x & 0x3f;

}

int bitCount\_standard(int x) {

int mask = 1,count=0;

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

if (x & mask) {

++count;

}

x >>= 1;

}

return count;

}

/\*(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) {

int l = ~0 << lowbit;

int h = ~(~0 << (highbit + 1));

return l & h;

}

int bitMask\_standard(int highbit, int lowbit) {

int x = 0, mask = 1;

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

if (i >= lowbit && i <= highbit) {

x |= mask;

}

mask <<= 1;

}

return x;

}

/\*(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) {

int res = x + y;

int x\_s = x >> 31;

int y\_s = y >> 31;

int res\_s = res >> 31;

int tag1 = !!(x\_s ^ y\_s);

int tag2 = !(x\_s ^ res\_s);

return !(tag1 | tag2);

}

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

int res = x + y;

if (x > 0 && y > 0 && res < 0 || x < 0 && y < 0 && res>0) {

return 1;

}

return 0;

}

/\*(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) {

int a = (x >> (n << 3)) & 0xff;

int b = (x >> (m << 3)) & 0xff;

x &= ~(0xff << (n << 3));

x &= ~(0xff << (m << 3));

x |= (a << (m << 3));

x |= (b << (n << 3));

return x;

}

/\*

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

}

\*/

int main() {

int choice,arg1,arg2,arg3,res;

while (true)

{

system("cls");

cout << "Select an number to run the function below!" << endl;

cout << "1.absVal 2.negate 3.bitAnd 4.bitOr" << endl;

cout << "5.bitXor 6.isTmax 7.bitCount 8.bitMask" << endl;

cout << "9.addOK 10.byteSwap" << endl;

cout << "Your choice( Enter 0 to exit):";

cin >> choice;

if (!choice) {

break;

}

switch (choice)

{

case 1:

cout << "Argument 1:";

cin >> arg1;

res=absVal(arg1);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 2:

cout << "Argument 1:";

cin >> arg1;

res = negate(arg1);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 3:

cout << "Argument 1-2:";

cin >> arg1 >> arg2;

res = bitAnd(arg1, arg2);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 4:

cout << "Argument 1-2:";

cin >> arg1 >> arg2;

res = bitOr(arg1, arg2);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 5:

cout << "Argument 1-2:";

cin >> arg1 >> arg2;

res = bitXor(arg1, arg2);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 6:

cout << "Argument 1:";

cin >> arg1;

res = isTmax(arg1);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 7:

cout << "Argument 1:";

cin >> arg1;

res = bitCount(arg1);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 8:

cout << "Highbit and lowbit:";

cin >> arg1 >> arg2;

res = bitMask(arg1, arg2);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 9:

cout << "Argument 1-2:";

cin >> arg1 >> arg2;

res = addOK(arg1, arg2);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

case 10:

cout << "x, n, m : ";

cin >> arg1 >> arg2>>arg3;

res = byteSwap(arg1, arg2,arg3);

cout << "Your result: " << res << endl;

Sleep(3000);

break;

default:

cout << "Please enter a valid number( 0 - 10 )!";

Sleep(3000);

break;

}

}

return 0;

}