

**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 ： CS2205**

**学 号 ： U202215510**

**姓 名 ： 徐新飏**

**指导教师 ： 李专**

**2024年 3 月 17 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**

题目已经定义了定义了结构体student来表示学生信息，包括姓名、年龄、成绩和备注，定义了两个结构体数组old\_s和new\_s，用于存放原始和解压后的学生信息。

struct数组由于存在空余空间，数据对齐等原因，导致结构体内部存在大量空间未被利用，因此，需要设计合适的压缩规则去完成数据压缩。

实验首先通过循环输入学生信息，包括姓名、年龄、成绩和备注，存储到old\_s数组中。

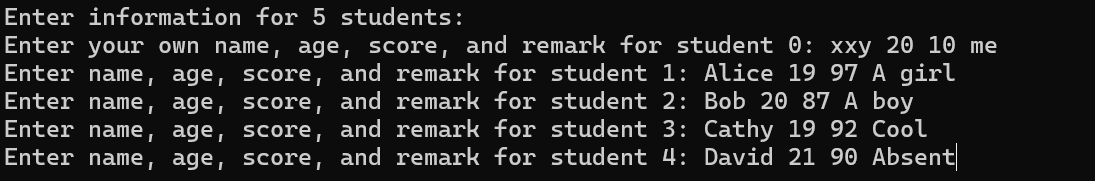


图1 输入信息

随后使用两个压缩函数pack\_student\_bytebybyte和pack\_student\_whole将学生信息压缩到字符数组message中。

其中pack\_student\_bytebybyte的设计思路是，利用for循环，从old\_s[0]的首地址开始，通过strlen来获取student.name的长度，将message的第一个字符作为标记位，该标记位的值等于name字符串的长度，随后将length元素空间用于存储姓名，由于提前存储了姓名的长度，所以最后不用存储’\0’,接下来的数据是short类型的age和float类型的score，分别占用两个和四个字节（注意short这里的字节是与存储顺序相反的，这里采用正序存储，在最后解码的时候反向读取即可），在message中存储后，与student.name的思路相同，额外使用三个标记位用来记录remark字符串的长度，同时依据已记录的长度读取相应长度的remark字符串，存储到message中。

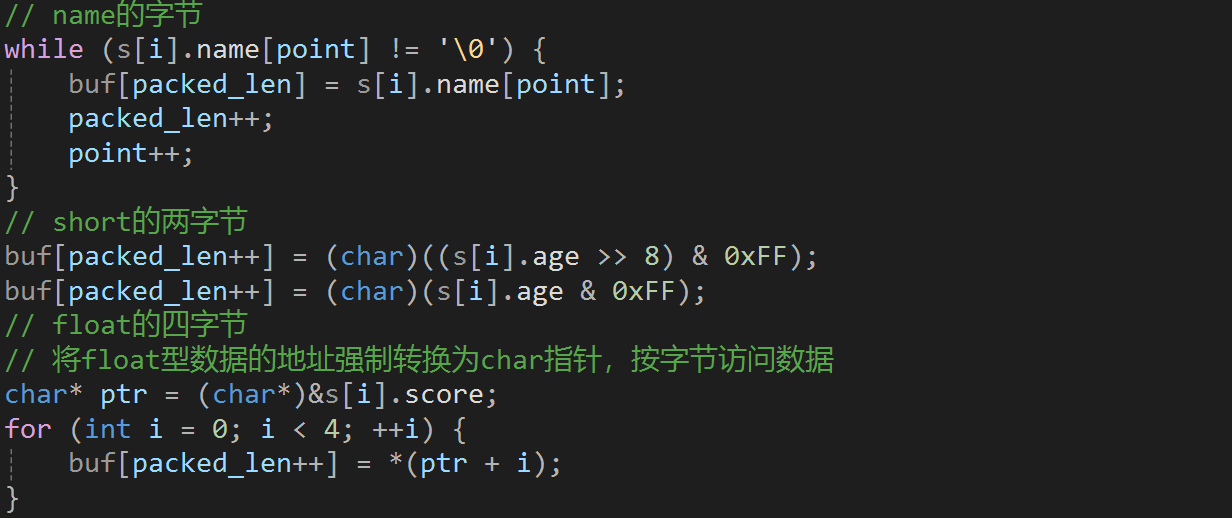


图2 pack\_student\_bytebybyte的定义

pack\_student\_whole的设计思路与pack\_student\_bytebybyte类似，只是使用了memcpy函数进行了对已经计算出的内存字节长度进行了整体拷贝，函数整体显得更为简洁。

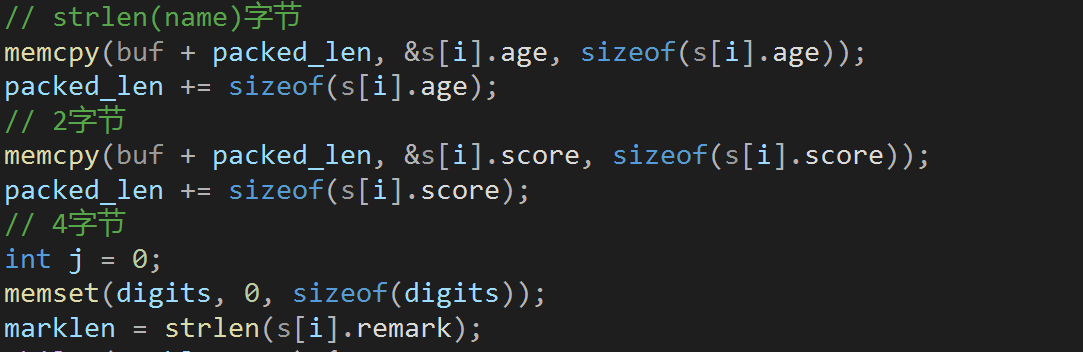


图3 pack\_student\_whole的定义

编写解压函数restore\_student，从压缩后的message中还原学生信息到new\_s数组中。

restore\_student的思路是，从头开始读取message数组,由于对每段student，message数组都记录了name和remark的长度，message读取到最开头的长度后，往后读取相同长度的字节，并存入new\_s数组相应的位置中，随后自然遇到下一个表示长度的数值，以此类推，直至读到message数组的结尾。

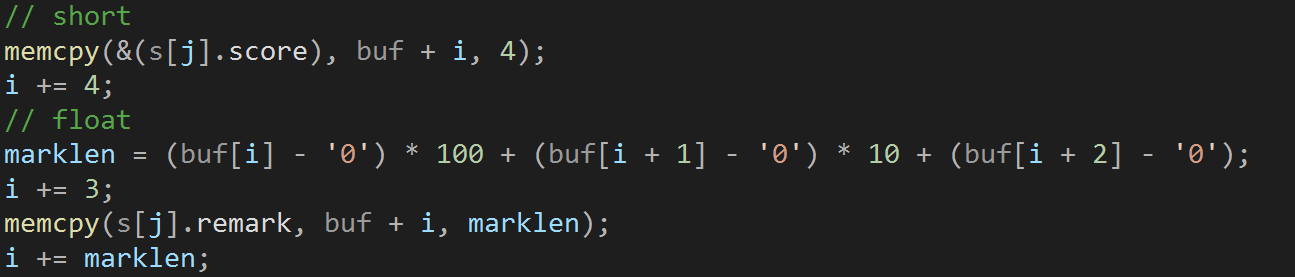


图4 restore\_student的定义

正如前面所提到的，使用pack\_student\_bytebybyte复制short型数据时两字节所处的位置相反，只需要在解码时特判一下反向复制即可。

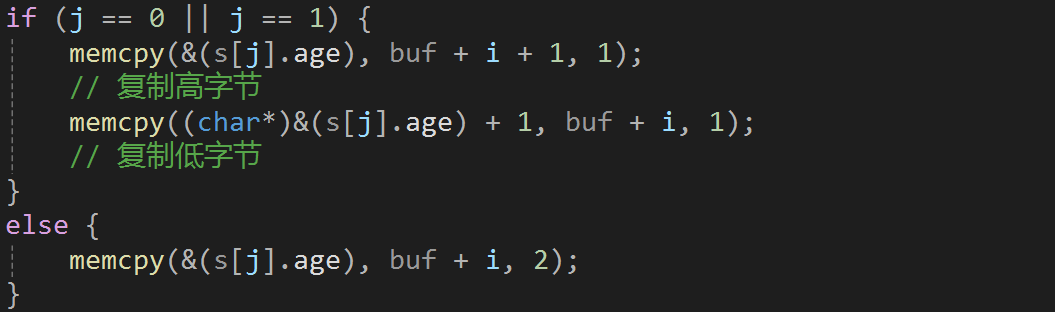


图5 restore\_student的特判部分

最后在输出控制台上的信息为:

输入的测试数据。

压缩前后数据的长度。

解压后的学生信息。

message前20个字节的内容。

其中第一个字节位name长度，为3，接下来三个为输入第一个学生的ASCII码,根据ASCII码表解码得到其为xxy，接下来两个元素为short型的age和float型的score，根据补码表示和IEEE754规则可以得出其表示20和10，接下来的三字节为remark的长度，为002（即表示长度为2），随后为remark的内容，通过ASCII码表解码得到其内容为me。随后的35即表示下一个student的名字长度为5。

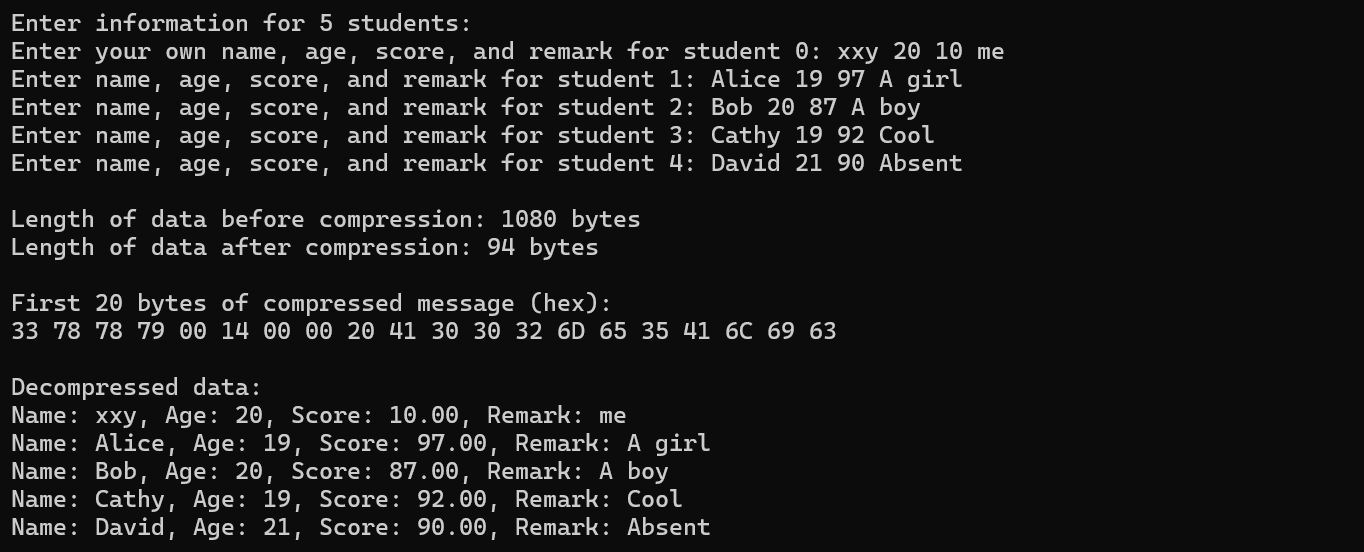
****

图6 输出控制台

与message通过内存查看的数值一致。

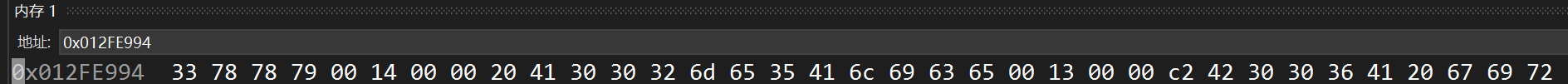


图7 在内存中查看message

验证第0个学生的成绩与计算是否一致：

计算过程：根据IEEE754规则，10的二进制表示为1010=1.010\*2^3。对正数，符号位为0，32位浮点数的指数位偏移量为127，那么阶码为130=10000010，尾数为010000000000000,最终将其表示为十六进制为41 20 00 00，计算机x86中为小端存储，故为00 00 20 41，可见，计算与实际结果一致。



图8 float各个部分编码

结构数组中元素的存放规律是按照结构体定义的顺序存放，即姓名、年龄、成绩和备注。字符串数组、short类型的数、float型的数分别按照其在结构体中的顺序存放，其中字符串数组为大端存储，short为大端存储，float为小端存储。

经过实验，该程序能够正确地实现数据的压缩、解压，并能够验证压缩前后数据的一致性。

**（2）任务 2 的算法思想、运行结果等记录**

1.绝对值函数absVal：

利用算术右移获取x的符号位作为掩码mask，如果x为负数，mask全为 1（即-1的补码），否则全为 0。根据mask对负数取反相当于~(x-1)，对正数相当于x，实现了取绝对值的功能。返回结果为 (x + mask) ^ mask。

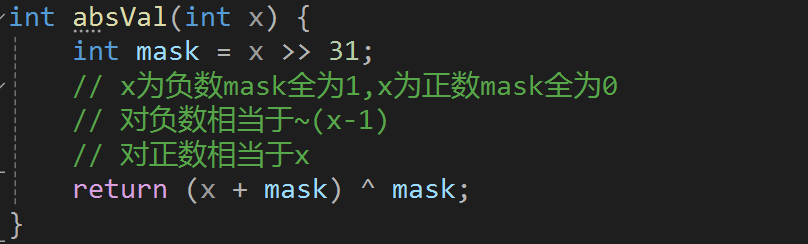


图9 absVal定义

2.取反函数negate：

~x=-1-x，故-x=~x+1，返回结果为~x + 1。

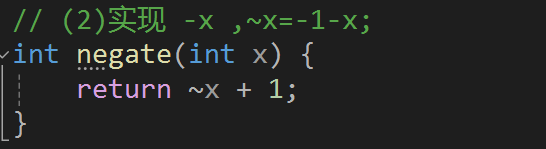


图10 negate定义

3.与运算函数bitAnd：

利用德摩根律将与运算转换为或非运算的形式，返回结果为~(~x | ~y)。

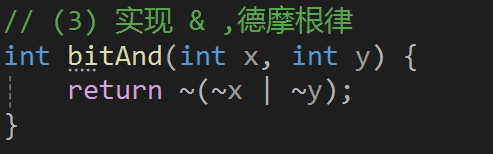


图11 bitAnd定义

4.或运算函数bitOr：

利用德摩根律将或运算转换为与非运算的形式，返回结果为~(~x & ~y)。

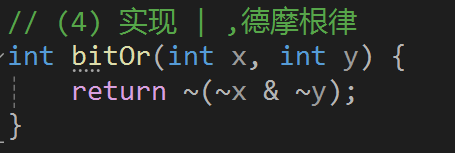


图12 bitOr定义

5.异或运算函数bitXor：

利用异或的逻辑展开和德摩根律，将异或运算转换为与非与非的形式，返回结果为~(~(~x & y) & ~(~y & x))。

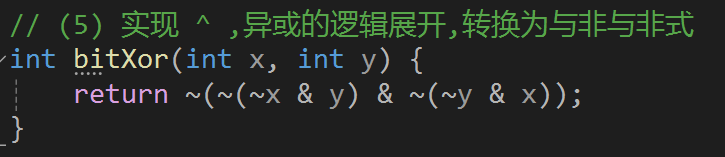


图13 bitXor定义

6.判断最大正整数函数isTmax：

注意到x 和 ~x+1 , 只有在0和Tmax时一样，那么只需要特判原数不等于0。将输入取反，然后加一，如果结果与原数相同且不等于 0，则说明原数是最大正整数，返回结果为!(x ^ (~x + 1)) & (!!x)。

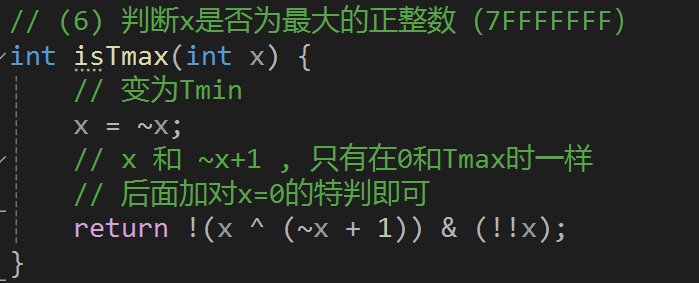


图14 isTmax定义

7.统计二进制表示中 1 的个数函数bitCount：

使用分治算法，利用与0x55555555，0x33333333，0x0f0f0f0f等常数逐步统计每2位，每4位，每8位，每16位和每32位中 1 的个数，利用分治再逐步累积的思想，返回结果即为最终累加得到的结果。最后的运算次数小于40次。

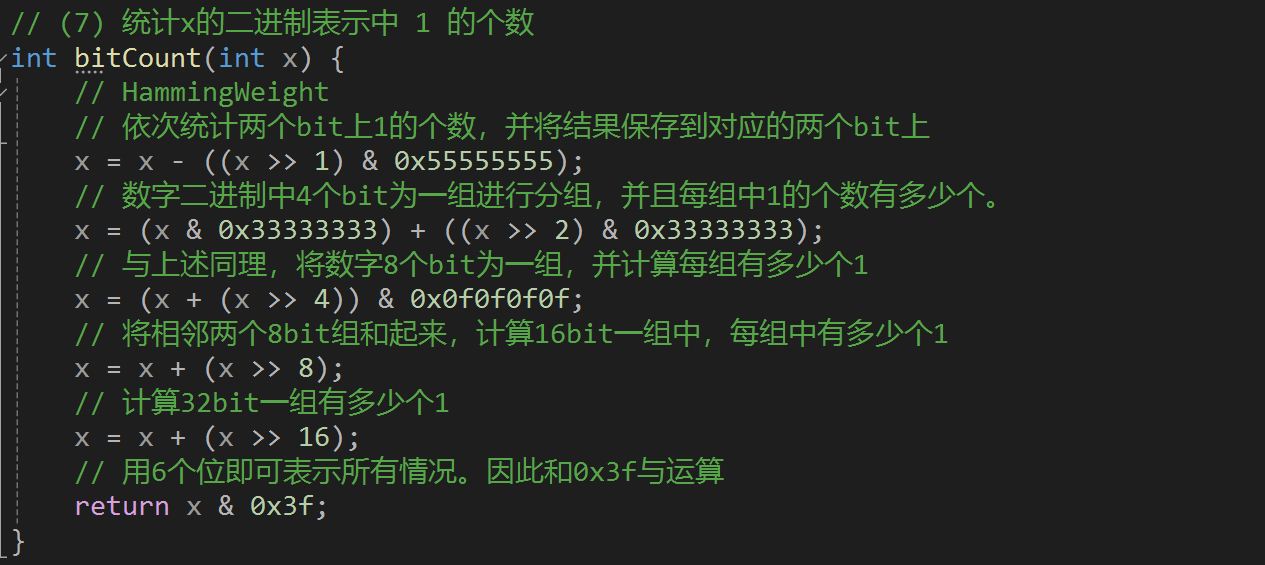


图15 bitCount定义

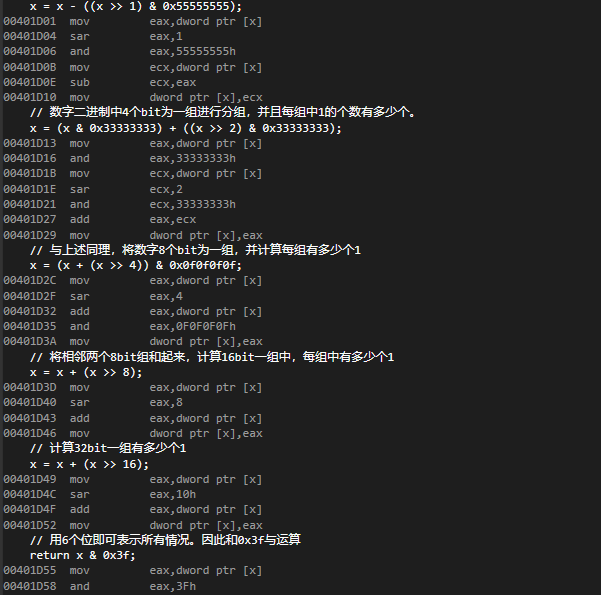


图16 bitCount反汇编代码

8.生成位函数bitMask：

通过对-1左移得到高位为1低位为0的串，控制两串之间的区别是1的个数为highbit + 1- lowbit。然后取异或来提取中间的一串1，生成从highbit+1到lowbit全为 1，其他位为 0 的数，返回结果为 (~0 << (highbit + 1)) ^ (~0 << lowbit)。最后的运算次数小于16次。

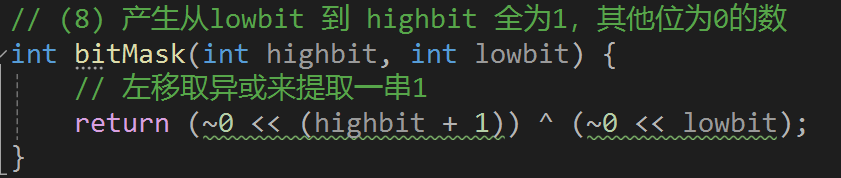


图17 bitMask定义

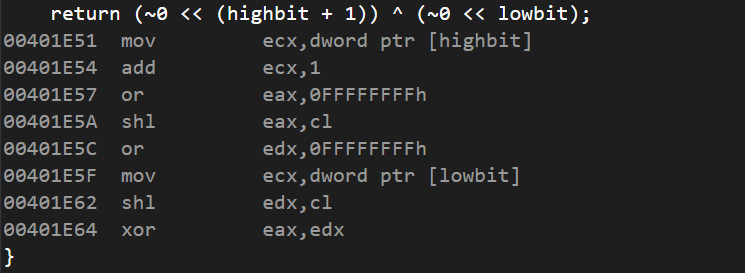


图18 bitMask反汇编代码

9.判断加法是否溢出函数addOK：

溢出只可能发生在两相同符号数相加的情况，如果两个数的和的符号位与两个加数的符号位不同，即个数符号位x,y=1,sum=0或x,y=0,sum=1的情况，最后通过两个!!来将输出结果确定为0或1，返回结果为!!(((x ^ sum) & (y ^ sum)) >> 31)。最后的运算次数小于20次。

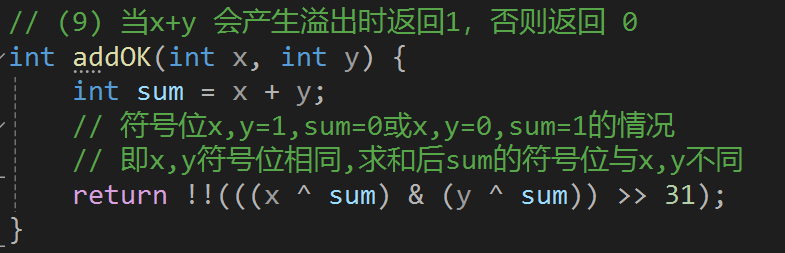


图19 addOK定义

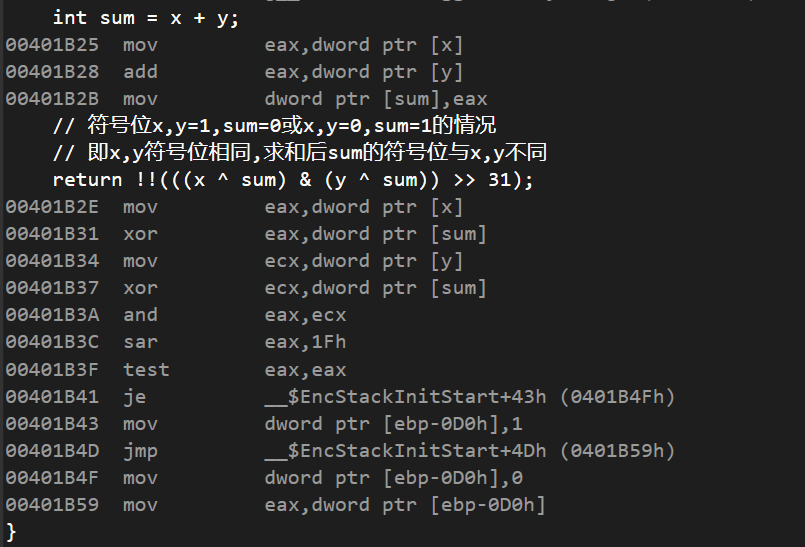


图20 addOK反汇编代码

10.字节交换函数byteSwap：

字节数乘8是位数，首先提取第n个字节和第m个字节的全部内容，然后制定第n和m个字节的掩码，用掩码取与后，将提取出来的第n个字节和第m个字节左移实现byteSwap。返回结果为 (x & ~((0xFF << (n << 3)) | (0xFF << (m << 3)))) | (((x >> (n << 3)) & 0xFF) << (m << 3)) | (((x >> (m << 3)) & 0xFF) << (n << 3))。最后的运算次数接近25次。

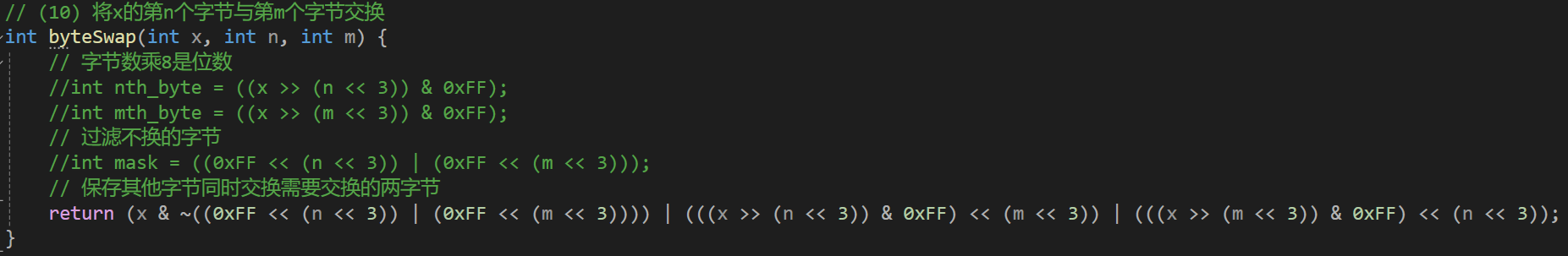


图21 byteSwap定义

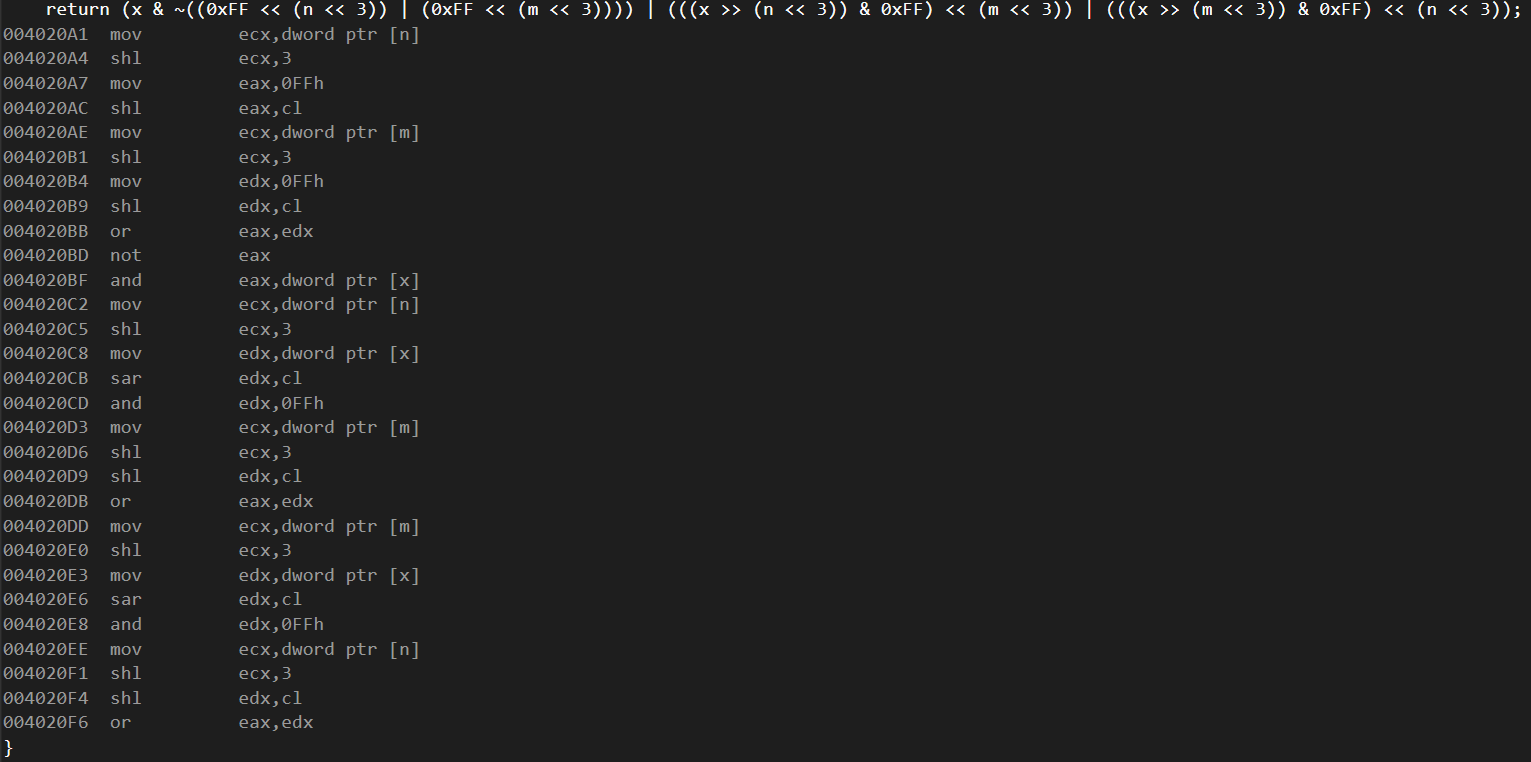


图22 byteSwap反汇编代码

在主函数中设置了对每个函数的测试，用户可以输入相应的操作，提供符合格式的数据后，程序会输出计算结果，并与标准函数的结果进行对比验证。

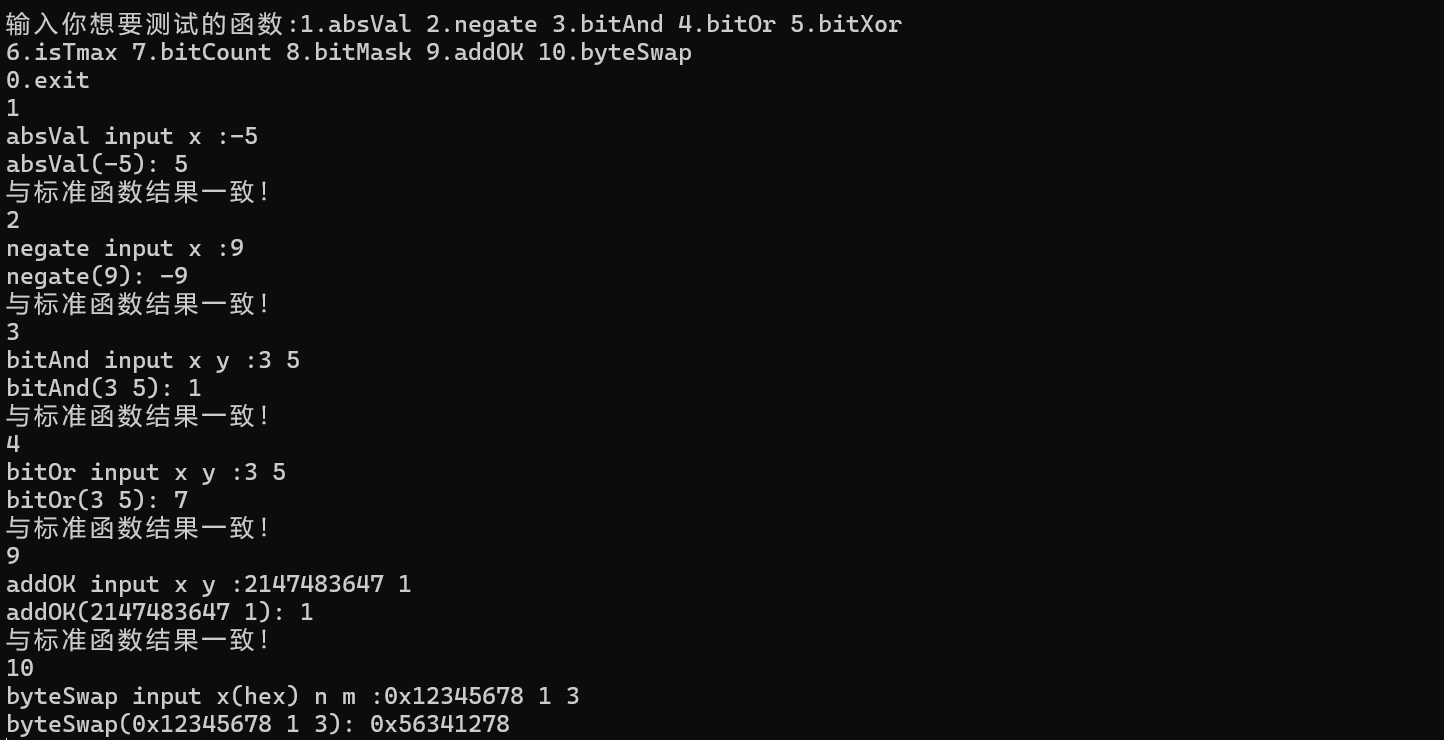


图23 函数测试界面

经过检验，该程序能够正确实现各个位运算函数的功能。

**四、体会**

完成以上两个实验后，我对程序开发平台(VS2022) 的用法有了更深入和更丰富的了解。从创建项目、编写代码、进行编译和链接，到在debug过程中对调试工具的使用，设置断点，单步执行，观察变量，查看内存窗口和反汇编窗口等，这使我能够更加高效地开发和调试程序。

在任务1中，我编写了数据存放的压缩与解压的程序，我深入了解了数据的存储原理和内存管理的相关知识，学会了如何有效地利用内存空间，提升了对数据结构和算法的理解，熟悉了处理不同数据类型的数据，包括int，short，char，float的存储位置，计算原理规则，存储大小等方面。

在任务2中，我编写了位运算程序，这个任务迫使我深入学习了位运算的原理和应用。我了解到计算机底层的运算原理，加深了对逻辑电路设计的理解，以及限制运算次数后，在提高程序效率和性能方面如何进行改进和优化。

这些经验对我今后的学习和工作都具有重要意义，使我更加自信地面对各种问题也让我更加热爱编程，激励我对计算机系统不断探索学习。

**五、源码**

实验任务 1、2 的源程序：

实验任务1源代码

1. #define \_CRT\_SECURE\_NO\_WARNINGS 1
2. #include <stdio.h>
3. #include <string.h>
4. #include <stdlib.h>
5. #define N 5
6. #define BUF\_SIZE 1500
7. struct student {
8. char name[8];
9. short age;
10. float score;
11. char remark[200];
12. };
13. const int N1 = 2;
14. const int N2 = 3;
15. int pack\_student\_bytebybyte(struct student\* s, int sno, char\* buf);
16. int pack\_student\_whole(struct student\* s, int sno, char\* buf);
17. int restore\_student(char\* buf, int len, struct student\* s);
18. int main() {
19. struct student old\_s[N], new\_s[N];
20. char message[BUF\_SIZE];
21. int i, packed\_len;
22. memset(old\_s, 0, sizeof(old\_s));
23. memset(message, 0, sizeof(message));
24. memset(new\_s, 0, sizeof(new\_s));
25. *// 输入学生信息*
26. printf("Enter information for %d students:\n", N);
27. for (i = 0; i < N; i++) {
28. *//对第0个姓名做特殊要求*
29. if (i == 0) {
30. printf("Enter your own name, age, score, and remark for student 0: ");
31. scanf("%s %hd %f %[^\n]", old\_s[i].name, &old\_s[i].age, &old\_s[i].score, old\_s[i].remark);
32. }
33. else {
34. printf("Enter name, age, score, and remark for student %d: ", i);
35. scanf("%s %hd %f %[^\n]", old\_s[i].name, &old\_s[i].age, &old\_s[i].score, old\_s[i].remark);
36. }
37. }
38. *// 压缩前存放数据的长度*
39. printf("\nLength of data before compression: %lu bytes\n", sizeof(old\_s));
40. *// 压缩调用pack\_student\_bytebybyte*
41. packed\_len = pack\_student\_bytebybyte(old\_s, N1, message);
42. *// 压缩调用pack\_student\_whole*
43. packed\_len += pack\_student\_whole(old\_s + 2, N2, message + packed\_len);
44. *// 打印压缩后存放数据的长度*
45. printf("Length of data after compression: %d bytes\n", packed\_len);
46. *// 输出message的前20个字节的内容（十六进制表示）*
47. printf("\nFirst 20 bytes of compressed message (hex):\n");
48. for (i = 0; i < 20 && i < packed\_len; i++) {
49. printf("%02X ", (unsigned char)message[i]);
50. }
51. printf("\n");
52. *// 解压数据到new\_s数组*
53. int num\_restored = restore\_student(message, packed\_len, new\_s);
54. *// 打印压缩前(old\_s)、解压后(new\_s)的结果*
55. printf("\nDecompressed data:\n");
56. for (i = 0; i < num\_restored; i++) {
57. printf("Name: %s, Age: %d, Score: %.2f, Remark: %s\n", new\_s[i].name, new\_s[i].age, new\_s[i].score, new\_s[i].remark);
58. }
59. *// 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*
60. printf("\nEncoding of score for student 0: ");
61. unsigned char\* score\_bytes = (unsigned char\*)&old\_s[0].score;
62. for (i = 0; i < sizeof(float); i++) {
63. printf("%02X ", score\_bytes[i]);
64. }
65. printf("\n");
66. return 0;
67. }
68. *//逐字节向message中写数据*
69. int pack\_student\_bytebybyte(struct student\* s, int sno, char\* buf) {
70. int i, packed\_len = 0;
71. int point = 0;
72. int digits[3];
73. *// 200 最多有 3 位数字*
74. int marklen = 0;
75. for (i = 0; i < sno; i++) {
76. point = 0;
77. buf[packed\_len] = '0' + (char)(strlen(s[i].name));
78. packed\_len++;
79. *// 额外扩充以为表示姓名压缩字节数*
80. *// name的字节*
81. while (s[i].name[point] != '\0') {
82. buf[packed\_len] = s[i].name[point];
83. packed\_len++;
84. point++;
85. }
86. *// short的两字节*
87. buf[packed\_len++] = (char)((s[i].age >> 8) & 0xFF);
88. buf[packed\_len++] = (char)(s[i].age & 0xFF);
89. *// float的四字节*
90. *// 将float型数据的地址强制转换为char指针，按字节访问数据*
91. char\* ptr = (char\*)&s[i].score;
92. for (int i = 0; i < 4; ++i) {
93. buf[packed\_len++] = \*(ptr + i);
94. }
95. int j = 0;
96. memset(digits, 0, sizeof(digits));
97. marklen = strlen(s[i].remark);
98. while (marklen > 0) {
99. digits[j++] = marklen % 10;
100. marklen /= 10;
101. }
102. buf[packed\_len] = '0' + (char)digits[2];
103. packed\_len++;
104. buf[packed\_len] = '0' + (char)digits[1];
105. packed\_len++;
106. buf[packed\_len] = '0' + (char)digits[0];
107. packed\_len++;
108. point = 0;
109. while (s[i].remark[point] != '\0') {
110. buf[packed\_len] = s[i].remark[point];
111. packed\_len++;
112. point++;
113. }
114. }
115. return packed\_len;
116. }
117. int pack\_student\_whole(struct student\* s, int sno, char\* buf) {
118. int i, packed\_len = 0;
119. int digits[3];
120. *// 200 最多有 3 位数字*
121. int marklen = 0;
122. for (i = 0; i < sno; i++) {
123. buf[packed\_len] = '0' + (char)(strlen(s[i].name));
124. packed\_len++;
125. *// 额外扩充以为表示姓名压缩字节数*
126. strcpy(buf + packed\_len, s[i].name);
127. packed\_len += strlen(s[i].name);
128. *// strlen(name)字节*
129. memcpy(buf + packed\_len, &s[i].age, sizeof(s[i].age));
130. packed\_len += sizeof(s[i].age);
131. *// 2字节*
132. memcpy(buf + packed\_len, &s[i].score, sizeof(s[i].score));
133. packed\_len += sizeof(s[i].score);
134. *// 4字节*
135. int j = 0;
136. memset(digits, 0, sizeof(digits));
137. marklen = strlen(s[i].remark);
138. while (marklen > 0) {
139. digits[j++] = marklen % 10;
140. marklen /= 10;
141. }
142. buf[packed\_len] = '0' + (char)digits[2];
143. packed\_len++;
144. buf[packed\_len] = '0' + (char)digits[1];
145. packed\_len++;
146. buf[packed\_len] = '0' + (char)digits[0];
147. packed\_len++;
148. strcpy(buf + packed\_len, s[i].remark);
149. packed\_len += strlen(s[i].remark);
150. }
151. return packed\_len;
152. }
153. int restore\_student(char\* buf, int len, struct student\* s) {
154. int i = 0;
155. *// i为buf的指针*
156. int j = 0;
157. int namelen = 0;
158. int marklen = 0;
159. for (j = 0; j < 5; j++) {
160. namelen = buf[i] - '0';
161. i++;
162. memcpy(s[j].name, buf + i, namelen);
163. i += namelen;
164. if (j == 0 || j == 1) {
165. memcpy(&(s[j].age), buf + i + 1, 1);
166. *// 复制高字节*
167. memcpy((char\*)&(s[j].age) + 1, buf + i, 1);
168. *// 复制低字节*
169. }
170. else {
171. memcpy(&(s[j].age), buf + i, 2);
172. }
173. i += 2;
174. *// short*
175. memcpy(&(s[j].score), buf + i, 4);
176. i += 4;
177. *// float*
178. marklen = (buf[i] - '0') \* 100 + (buf[i + 1] - '0') \* 10 + (buf[i + 2] - '0');
179. i += 3;
180. memcpy(s[j].remark, buf + i, marklen);
181. i += marklen;
182. }
183. return j;
184. }

实验任务2源代码：

1. #define \_CRT\_SECURE\_NO\_WARNINGS 1
2. #include <stdio.h>
3. *// (1)返回x的绝对值 ,分类讨论*
4. int absVal(int x) {
5. int mask = x >> 31;
6. *// x为负数mask全为1,x为正数mask全为0*
7. *// 对负数相当于~(x-1)*
8. *// 对正数相当于x*
9. return (x + mask) ^ mask;
10. }
11. int absVal\_standard(int x)
12. { return (x < 0) ? -x : x; }
13. *// (2)实现 -x ,~x=-1-x;*
14. int negate(int x) {
15. return ~x + 1;
16. }
17. int negate\_standard(int x) { return -x; }
18. *// (3) 实现 & ,德摩根律*
19. int bitAnd(int x, int y) {
20. return ~(~x | ~y);
21. }
22. int bitAnd\_standard(int x, int y) { return x & y; }
23. *// (4) 实现 | ,德摩根律*
24. int bitOr(int x, int y) {
25. return ~(~x & ~y);
26. }
27. int bitOr\_standard(int x, int y) { return x | y; }
28. *// (5) 实现 ^ ,异或的逻辑展开,转换为与非与非式*
29. int bitXor(int x, int y) {
30. return ~(~(~x & y) & ~(~y & x));
31. }
32. int bitXor\_standard(int x, int y) { return x ^ y; }
33. *// (6) 判断x是否为最大的正整数（7FFFFFFF）*
34. int isTmax(int x) {
35. *// 变为Tmin*
36. x = ~x;
37. *// x 和 ~x+1 , 只有在0和Tmax时一样*
38. *// 后面加对x=0的特判即可*
39. return !(x ^ (~x + 1)) & (!!x);
40. }
41. int isTmax\_standard(int x) {
42. return x == 0x7FFFFFFF;
43. }
44. *// (7) 统计x的二进制表示中 1 的个数*
45. int bitCount(int x) {
46. *// HammingWeight*
47. *// 依次统计两个bit上1的个数，并将结果保存到对应的两个bit上*
48. x = x - ((x >> 1) & 0x55555555);
49. *// 数字二进制中4个bit为一组进行分组，并且每组中1的个数有多少个。*
50. x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
51. *// 与上述同理，将数字8个bit为一组，并计算每组有多少个1*
52. x = (x + (x >> 4)) & 0x0f0f0f0f;
53. *// 将相邻两个8bit组和起来，计算16bit一组中，每组中有多少个1*
54. x = x + (x >> 8);
55. *// 计算32bit一组有多少个1*
56. x = x + (x >> 16);
57. *// 用6个位即可表示所有情况。因此和0x3f与运算*
58. return x & 0x3f;
59. }
60. int bitCount\_standard(int x) {
61. int result = 0;
62. int i;
63. for (i = 0; i < 32; i++)
64. result += (x >> i) & 0x1;
65. return result;
66. }
67. *// (8) 产生从lowbit 到 highbit 全为1，其他位为0的数*
68. int bitMask(int highbit, int lowbit) {
69. *// 左移取异或来提取一串1*
70. return (~0 << (highbit + 1)) ^ (~0 << lowbit);
71. }
72. int bitMask\_standard(int highbit, int lowbit)
73. {
74. int result = 0;
75. int i;
76. for (i = lowbit; i <= highbit; i++)
77. result |= 1 << i;
78. return result;
79. }
80. *// (9) 当x+y 会产生溢出时返回1，否则返回 0*
81. int addOK(int x, int y) {
82. int sum = x + y;
83. *// 符号位x,y=1,s=0或s,y=0,s=1的情况*
84. *// 即x,y符号位相同,求和后sum的符号位与x,y不同*
85. return !!(((x ^ sum) & (y ^ sum)) >> 31);
86. }
87. int addOK\_standard(int x, int y)
88. {
89. long long lsum = (long long)x + y;
90. if (lsum == (int)lsum) {
91. return 0;
92. }
93. else return 1;
94. }
95. *// (10) 将x的第n个字节与第m个字节交换*
96. int byteSwap(int x, int n, int m) {
97. *// 字节数乘8是位数*
98. *//int nth\_byte = ((x >> (n << 3)) & 0xFF);*
99. *//int mth\_byte = ((x >> (m << 3)) & 0xFF);*
100. *// 过滤不换的字节*
101. *//int mask = ((0xFF << (n << 3)) | (0xFF << (m << 3)));*
102. *// 保存其他字节同时交换需要交换的两字节*
103. return (x & ~((0xFF << (n << 3)) | (0xFF << (m << 3)))) | (((x >> (n << 3)) & 0xFF) << (m << 3)) | (((x >> (m << 3)) & 0xFF) << (n << 3));
104. }
105. int byteSwap\_standard(int x, int n, int m)
106. {
107. unsigned int nmask, mmask;
108. switch (n) {
109. case 0:
110. nmask = x & 0xFF;
111. x &= 0xFFFFFF00;
112. break;
113. case 1:
114. nmask = (x & 0xFF00) >> 8;
115. x &= 0xFFFF00FF;
116. break;
117. case 2:
118. nmask = (x & 0xFF0000) >> 16;
119. x &= 0xFF00FFFF;
120. break;
121. default:
122. nmask = ((unsigned int)(x & 0xFF000000)) >> 24;
123. x &= 0x00FFFFFF;
124. break;
125. }
126. switch (m) {
127. case 0:
128. mmask = x & 0xFF;
129. x &= 0xFFFFFF00;
130. break;
131. case 1:
132. mmask = (x & 0xFF00) >> 8;
133. x &= 0xFFFF00FF;
134. break;
135. case 2:
136. mmask = (x & 0xFF0000) >> 16;
137. x &= 0xFF00FFFF;
138. break;
139. default:
140. mmask = ((unsigned int)(x & 0xFF000000)) >> 24;
141. x &= 0x00FFFFFF;
142. break;
143. }
144. nmask <<= 8 \* m;
145. mmask <<= 8 \* n;
146. return x | nmask | mmask;
147. }
148. int main() {
149. int x, y, n, m;
150. int choice;
151. printf("输入你想要测试的函数:");
152. printf("1.absVal 2.negate 3.bitAnd 4.bitOr 5.bitXor\n"
153. "6.isTmax 7.bitCount 8.bitMask 9.addOK 10.byteSwap\n");
154. printf("0.exit\n");
155. while (scanf("%d", &choice) && choice != 0) {
156. switch (choice)
157. {
158. case 0: {
159. return 0;
160. }
161. case 1: {
162. printf("absVal input x :");
163. scanf("%d", &x);
164. printf("absVal(%d): %d\n", x, absVal(x));
165. if (absVal(x) == absVal\_standard(x)) {
166. printf("与标准函数结果一致！\n");
167. }
168. else {
169. printf("与标准函数结果不一致！\n");
170. }
171. }
172. break;
173. case 2: {
174. printf("negate input x :");
175. scanf("%d", &x);
176. printf("negate(%d): %d\n", x, negate(x));
177. if (negate(x) == negate\_standard(x)) {
178. printf("与标准函数结果一致！\n");
179. }
180. else {
181. printf("与标准函数结果不一致！\n");
182. }
183. }
184. break;
185. case 3: {
186. printf("bitAnd input x y :");
187. scanf("%d %d", &x, &y);
188. printf("bitAnd(%d %d): %d\n", x, y, bitAnd(x, y));
189. if (bitAnd(x, y) == bitAnd\_standard(x, y)) {
190. printf("与标准函数结果一致！\n");
191. }
192. else {
193. printf("与标准函数结果不一致！\n");
194. }
195. }
196. break;
197. case 4: {
198. printf("bitOr input x y :");
199. scanf("%d %d", &x, &y);
200. printf("bitOr(%d %d): %d\n", x, y, bitOr(x, y));
201. if (bitOr(x, y) == bitOr\_standard(x, y)) {
202. printf("与标准函数结果一致！\n");
203. }
204. else {
205. printf("与标准函数结果不一致！\n");
206. }
207. }
208. break;
209. case 5: {
210. printf("bitXor input x y :");
211. scanf("%d %d", &x, &y);
212. printf("bitXor(%d %d): %d\n", x, y, bitXor(x, y));
213. if (bitXor(x, y) == bitXor\_standard(x, y)) {
214. printf("与标准函数结果一致！\n");
215. }
216. else {
217. printf("与标准函数结果不一致！\n");
218. }
219. }
220. break;
221. case 6: {*//2147483647*
222. printf("isTmax input x :");
223. scanf("%d", &x);
224. printf("isTmax(%d): %d\n", x, isTmax(x));
225. if (isTmax(x) == isTmax\_standard(x)) {
226. printf("与标准函数结果一致！\n");
227. }
228. else {
229. printf("与标准函数结果不一致！\n");
230. }
231. }
232. break;
233. case 7: {
234. printf("bitCount input x :");
235. scanf("%d", &x);
236. printf("bitCount(%d): %d\n", x, bitCount(x));
237. if (bitCount(x) == bitCount\_standard(x)) {
238. printf("与标准函数结果一致！\n");
239. }
240. else {
241. printf("与标准函数结果不一致！\n");
242. }
243. }
244. break;
245. case 8: {
246. printf("bitMask input highbit lowbit :");
247. scanf("%d %d", &x, &y);
248. printf("bitMask(%d %d): 0x%X\n", x, y, bitMask(x, y));
249. if (bitMask(x, y) == bitMask\_standard(x, y)) {
250. printf("与标准函数结果一致！\n");
251. }
252. else {
253. printf("与标准函数结果不一致！\n");
254. }
255. }
256. break;
257. case 9: {*//2147483647*
258. printf("addOK input x y :");
259. scanf("%d %d", &x, &y);
260. printf("addOK(%d %d): %d\n", x, y, addOK(x, y));
261. if (addOK(x, y) == addOK\_standard(x, y)) {
262. printf("与标准函数结果一致！\n");
263. }
264. else {
265. printf("与标准函数结果不一致！\n");
266. }
267. }
268. break;
269. case 10: {
270. printf("byteSwap input x(hex) n m :");
271. scanf("%x %d %d", &x, &n, &m);
272. printf("byteSwap(0x%X %d %d): 0x%X\n", x, n, m, byteSwap(x, n, m));
273. if (byteSwap(x, n, m) == byteSwap\_standard(x, n, m)) {
274. printf("与标准函数结果一致！\n");
275. }
276. else {
277. printf("与标准函数结果不一致！\n");
278. }
279. }
280. break;
281. default: {
282. printf("请输入合法的数字(1~10)!\n");
283. }
284. break;
285. }
286. }
287. return 0;
288. }