# 实验1 表达式和标准输入与输出实验

## 1.1 实验目的

（1）熟练掌握各种运算符的运算功能，操作数的类型，运算结果的类型及运算过程中的类型转换，重点是C语言特有的运算符，例如位运算符，问号运算符，逗号运算符等；熟记运算符的优先级和结合性；

（2）掌握getchar, putchar, scanf 和printf 函数的用法。

（3）掌握简单C程序（顺序结构程序）的编写方法。

## 1.2 实验内容

**1.2.1 源程序改错题**

下面给出了一个简单C语言程序例程，用来完成以下工作：

（1）输入华氏温度f，将它转换成摄氏温度c后输出；

（2）输入圆的半径值ｒ，计算并输出圆的面积ｓ；

（3）输入短整数ｋ、ｐ，将ｋ的高字节作为结果的低字节，ｐ的高字节作为结果的高字节，拼成一个新的整数，然后输出；

在这个例子程序中存在若干语法和逻辑错误。要求在计算机上对这个例子程序进行调试修改，使之能够正确完成指定任务。

1 #include<stdio.h>

2 #define PI 3.14159;

3 intmain( void )

4 {

5 int f ;

6 short p, k ;

7 double c , r , s ;

8 /\* for task 1 \*/

9 printf(“Input Fahrenheit:” ) ;

10 scanf(“%d”, f ) ;

11 c = 5/9\*(f-32) ;

12 printf( “ \n %d (F) = %.2f (C)\n\n ”, f, c ) ;

13 /\* for task 2 \*/

14 printf("input the radius r:");

15 scanf("%f", &r);

16 s = PI \* r \* r;

17 printf("\nThe acreage is %.2f\n\n",&s);

18 /\* for task 3 \*/

19 printf("input hex int k, p :");

20 scanf("%x %x", &k, &p );

21 newint = (p&0xff00)|(k&0xff00)<<8;

22 printf("new int = %x\n\n",newint);

23 return 0;

}

解答:

1. 错误修改:
2. 第2行的符号常量定义后不能有分号，正确形式为：

#define PI 3.14159

1. 第3行的函数原型拼写错误，voidmain中间应有空格，正确形式为：

void main(void)

1. 第5行未定义task 3中将要用到的newint变量，正确形式为：

int f ,newint;

1. 第9行的双引号应使用"，正确形式为：

printf("Input Fahrenheit: ") ;

1. 第10行的变量前应加上&以取地址，双引号应使用"，正确形式为：

scanf("%d", &f ) ;

1. 第11行整型数据进行“/”运算会进行取整，应使用浮点型数据5.0等，正确形式为：

c=5.0/9.0\*(f-32);

1. 第12行的双引号应使用"，正确形式为：

printf( " \n %d (F) = %.2f (C)\n\n ", f, c ) ;

1. 第15行的转换字符使用错误，对应double型的r，应使用转换字符lf，正确形式为：

scanf("%lf", &r);

1. 第17行的数据项s前不能加上取地址操作，正确形式为：

printf("\nThe acreage is %.2f \n",s);

1. 第21行的表达式逻辑错误，k的高字节应向右移动，正确形式为：

newint=(p&0xff00)|((k&0xff00)>>8);

错误修改后源程序:

#include<stdio.h>

#define PI 3.14159

void main( void )

{

int f ,newint;

short p, k ;

double c , r , s ;

/\* for task 1 \*/

printf("Input Fahrenheit:" ) ;

scanf("%d", &f ) ;

c = 5.0/9.0\*(f-32) ;

printf( " \n %d (F) = %.2f (C)\n\n ", f, c ) ;

/\* for task 2 \*/

printf("input the radius r:");

scanf("%lf", &r);

s = PI \* r \* r;

printf("\nThe acreage is %.2f\n\n",s);

/\* for task 3 \*/

printf("input hex int k, p :");

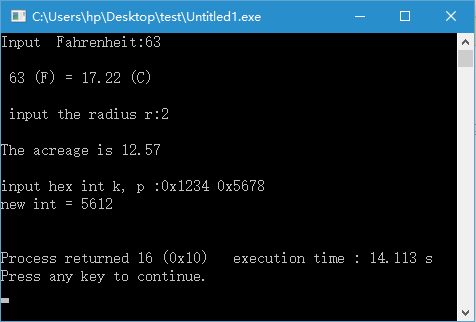
scanf("%x %x", &k, &p );

newint = (p&0xff00)|(k&0xff00)>>8;

printf("new int = %x\n\n",newint);

}

1. 错误修改后运行结果:



**1.2.2 源程序修改替换题**

下面的程序利用常用的中间变量法实现两数交换，请改用不用第三个变量的交换法实现。

#include<stdio.h>

void main( )

{

int a, b, t;

printf(“Input two integers:”);

scanf(“%d %d”,&a,&b);

t=a, a=b, b=t;

prinf(“\na=%d,b=%d”,a,b);

}

解答：

修改下划线处的语句为：

a=a+b; b=a-b; a=a-b;

替换后的程序如下所示：

#include<stdio.h>

void main( )

{

int a,b;

printf("Input two integers:");

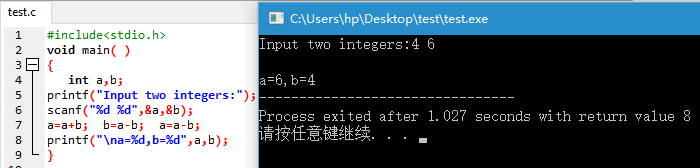
scanf("%d %d",&a,&b);

a=a+b; b=a-b; a=a-b;

printf("\na=%d,b=%d",a,b);

}

运行截图：



**1.2.3 程序设计**

**（1）**编写一个程序，输入无符号短整数x，ｍ，ｎ（0 ≤ｍ≤ 15, 1 ≤ ｎ≤ 16-ｍ）,取出x从第ｍ位开始向左的ｎ位（ｍ从右至左编号为0～15），并使其向左端（第15位）靠齐。

解答：

1. 解题思路：
2. 提示并输入x、m、n；
3. 判断m、n是否符合要求：如果0 ≤ m ≤ 15且1 ≤ n ≤ 16-m，转2.1，否则转3；
   1. 构建逻辑尺mask=0xFFFF，分别左移m位、右移(16-m-n)位，即mask1=mask<<(m)， mask2=mask>>(16-m-n)；
   2. choosemask=mask1&mask2，将上述移位后逻辑尺按位与（&），产生新的从第m位开始向左n位均为1的逻辑尺choosemask；
   3. p=choosemask & x，得到x从m位开始向左n位；
   4. 将p左移(16-m-n)位，输出结果并转4。
4. 显示输入错误信息；
5. 结束。
6. 程序清单：

#include<stdio.h>

int main()

{

unsigned x,m,n;

printf("please input x、m(0~15)、n(1~16-m):\n");

scanf("%u%u%u",&x,&m,&n);

if(m>=0&&m<=15&&n>=1&&n<=16-m)//判断m、n是值是否合理

{

long mask=0xFFFF;

long mask1=mask<<(m);

long mask2=mask>>(16-m-n);

long choosemask=mask1&mask2;//构建m开始到第n位为1的逻辑尺

unsigned p;

p=choosemask&x;

p=p<<(16-m-n);

printf("%u\n",p);

}

else

printf("wrong input!\n");

return 0;

}

1. 测试
2. 测试数据：

为方便分析测试结果，选择x分别为32768、1，同时尽量选取两端的字符。如表1-1所示。

表1-1 编程题3.(1)的测试数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | | | 理 论 结 果 |
| x | M | n |
| 用例1 | 32768 | 15 | 1 | 计算结果 1000 0000 0000 0000 即32768 |
| 用例2 | 1 | 0 | 1 | 计算结果 1000 0000 0000 0000 即32768 |
| 用例3 | 1 | 0 | 2 | 计算结果 0100 0000 0000 0000 即16384 |

1. 对应测试测试用例1的运行结果如图1-1所示。

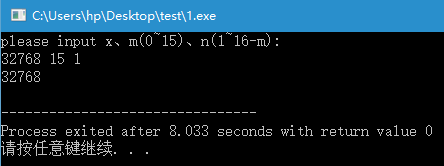


图1-1 编程题3.(1)的测试用例一的运行结果

1. 对应测试测试用例2的运行结果如图1-2所示。

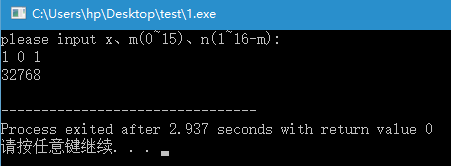


图1-2 编程题3.(1)的测试用例二的运行结果

1. 对应测试测试用例3的运行结果如图1-3所示。

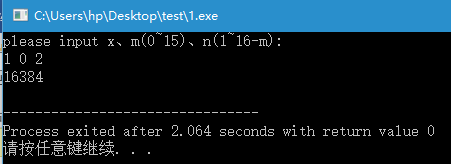


图1-3 编程题3.(1)的测试用例三的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

（2）IP地址通常是4个用句点分隔的小整数，如32.55.1.102。这些地址在机器中用无符号长整形表示。编写一个程序，以机器存储的形式读入一个32位的互联网IP地址，对其译码，然后用常见的句点分隔的4部分的形式输出。

解答：

1. 解题思路：
2. 提示并输入32位无符号长整形N；
3. 构建逻辑尺并用数组存储，即mask[4] = { 0x000000FF , 0x0000FF00 , 0x00FF0000 , 0xFF000000}，逻辑尺用以将长整形N的对应部分还原为被句点分隔的小整数；
4. 分别将N的对应部分用逻辑尺筛出，并通过定义整形i，使四个小整数部分分别向右移动相应位数，即

int i=0;

long dec =(N&mask[i])>>(i\*8);

1. 使用for循环依次输出四个小整数，并在前三个小整数后输出句点”.”以达成格式要求，即

for(; i<4 ; i++ )

{

long dec =(N&mask[i])>>(i\*8);

char a[100];

sprintf(a,"%ld",dec);

printf("%s",a);

if(i!=3)

printf(".");

}

1. 结束。
2. 程序清单：

#include<stdio.h>

int main(void)

{

unsigned long N;

scanf("%ul",&N) ;

unsigned long mask[4] = {0x000000FF , 0x0000FF00 , 0x00FF0000 , 0xFF000000 };

int i=0;

for(; i<4 ; i++ )

{

long dec =(N&mask[i])>>(i\*8);

char a[100];

sprintf(a,"%ld",dec);

printf("%s",a);

if(i!=3)

printf(".");

}

printf("\n");

}

1. 测试
2. 测试数据

为便于分析结果，选择N分别为134744072、16843009等特殊数字，另外输入一组较一般的数据16885952。如表1-2所示。

1. 表1-2 编程题3.(2)的测试数据

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |
| N |
| 用例1 | 134744072 | 输出结果 8.8.8.8 |
| 用例2 | 16885952 | 输出结果 192.168.1.1 |
| 用例3 | 16843009 | 输出结果 1.1.1.1 |

1. 对应测试测试用例1的运行结果如图1-4所示。

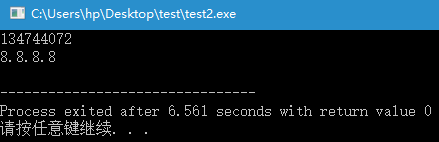


图1-4 编程题3.(2)的测试用例一的运行结果

1. 对应测试测试用例2的运行结果如图1-5所示。

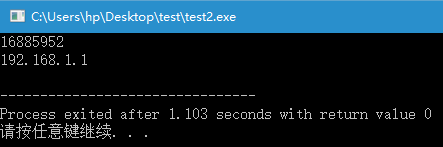


图1-5 编程题3.(2)的测试用例二的运行结果

1. 对应测试测试用例3的运行结果如图1-6所示。

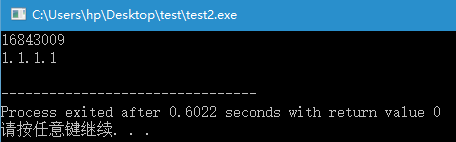


图1-6 编程题3.(2)的测试用例三的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

## 1.3 自设题

**（1）**自设实验题目：实数型的浮点表示误差的简单验证

**（2）**实验目的：通过设计实验程序，理解实数的浮点数表示上的误差，浮点数不宜做等值比较。

**（3）**实验程序：

#include "stdio.h"

int main()

{

float a=10.33;

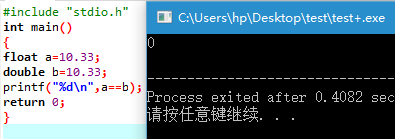
double b=10.33;

printf("%d\n",a==b);

return 0;

}

**（4）**实验用例：如图



**（5）**实验结论：由于内存中实数的浮点数的表示存在误差，浮点数不宜做等值比较，如用例中a与b不等值。

## 1.4 实验小结

在实验中主要遇到的问题有：

1. 在1.2.1源程序改错的task 1中一开始没有发现第十行中的错误，即变量f未取地址，直接运行后编译器未报错，但运行后会未响应。

后在与同学的讨论中发现问题并解决。

1. 在1.2.2源程序修改替换时，第一次遇见这类题型并无思路。

经过思考后想出解决方案。

1. 在解决问题1.2.3程序设计（1）时，未考虑m、n的取值判断。

检查程序时发现问题并处理。

体会：

1. 在编写程序时需严加注意格式方面的问题，如scanf时未加取地址符号&，这类错误编译器可能并不会提醒，若在大工程中则很难被发现。建议以后完成一部分调试一部分，或编写函数使程序更易调试，或使用VS中的单步调试处理错误；
2. 解决同一个问题的程序远不止几种，各个方案间的差距可能会很大，采用适当的方式可如1.2.2替换源程序一样减少变量的定义，对同一个问题采用其他思考方式更可能大幅优化程序；
3. 编程时重心虽然在通过程序的主干部分解决问题，但前提的判断等系列问题也是需要注意而且不可缺少的，如1.2.3（1）中源程序头部对m、n取值的判断，编程时应保证程序的可移植性与稳定性。