C语言第五周作业-2

阶乘求解

> 【问题描述】3.2.1 输入一个小于等于10的数n，求n!。
> 【样例输入】 `Enter a number:5`
> 【样例输出】 `5!=120`
> 【样例说明】本题输出结束后没有换行符。

- 首先我们要知道n的阶乘计算方法：$!n = 1 \times 2 \times 3 \cdots \times n$
  - 显然我们只需要写一个 `for` 循环就ok了
  - 只不过要注意累乘的初始值应该定为 `1`

```c
#include <stdio.h>
int factorial(int n){
    int res = 1;
    for (int i = 1; i <= n; i++)
        res *= i;
    return res;
}
int main()
{
    int n;
    printf("Enter a number:");
    scanf("%d",&n);
    printf("%d!=%d",n,factorial(n));
    return 0;
}
```

- 这个可以看到一个自定义函数 `factorial(n)` ，目前这个进度下咱们没学，其实把它放到代码里面就行。

```c
#include <stdio.h>
int main()
{
    int n, res = 1;
    printf("Enter a number:");
    scanf("%d",&n);
    for (int i = 1; i <= n; i++)
        res *= i;
    printf("%d!=%d",n,res);
    return 0;
}
```

当然这样看起来会简洁很多。

奇偶数统计

> 【问题描述】编写一个程序。该程序读取整数，直到输入为0时终止。输入终止后，报告输入的偶数个数、偶数平均值，输入的奇数个数，奇数平均值。
> 【输入形式】输入整数
> 【输出形式】偶数个数、偶数平均值（浮点数，保留两位小数），输入的奇数个数，奇数平均值（浮点数，保留两位小数）
> 【输入样例】`1 2 3 4 5 6 0`
> 【输出样例】
> `Number of even:3;Average of even:4.00`
> `Number of odd:3;Average of odd:3.00`（此处输出结束有换行符，冒号和分号后无空格。）

- 很简单的想法，我们一个个读入然后进行判断计数
    - 奇数对2取余不等于0，偶数对2取余等于0（奇偶数判断条件）

```c
#include <stdio.h>
int main()
{
    int a,sum_o = 0,sum_e = 0,i_o = 0,i_e = 0;//i_e记录偶数个数，i_o记录奇数个数
    while(scanf("%d",&a) && a!=0) {
        if(a % 2 != 0) {
            sum_o += a;
            i_o ++;
        }
        else {
            sum_e += a;
            i_e ++;
        }
    }
    printf("Number of even:%d;Average of even:%.2f\nNumber of odd:%d;Average of odd:%.2f",i_e,(double)sum_e/(double)i_e,i_o,(double)sum_o/(double)i_o);//这里分成两行写成两条printf也行
    return 0;
}
```

素数判断

> 【问题描述】判断输入的某个数是否为素数。若是，输出YES，否则输出NO。
> 【输入形式】一个数
> 【输出形式】YES 或者 NO，输出结束无换行符。
> 【样例输入】 `2`
> 【样例输出】 `YES`
> 【样例说明】本题输出结束后没有换行符。

- 我们首先来了解什么是 `素数` ： **质数（Prime number）**，又称素数，指在**大于1的自然数**中，除了1和该数自身外，无法被其他自然数整除的数（也可定义为只有1与该数本身两个正因数的数）。
- 既然是整除，那么我们就可以通过取余运算结果是否为0来判断是否能够整除
- 另外我们只用判断$1 \rightarrow \sqrt{x}$这个范围内的数就好了，减少循环量

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int n;
    scanf("%d",&n);
    for(int i = 2;i <= sqrt((double)n); i++){
        if(n%i == 0){
            printf("NO");
            return 0;
        }
    }
    printf("YES");
    return 0;
}
```

求最大公约数与最小公倍数

> 【问题描述】3.2.4 输入两个正整数m和n，求m和n的最大公约数及最小公倍数。
> 【输入形式】从键盘输入两个正整数。
> 【输入输出样例1】
> **Input m,n:6 15**
> **6和15的最大公约数:3**
> **6和15的最小公倍数:30**
> 【输入输出样例2】
> **Input m,n:-2 3**
> **Input again!**
> **15 8**
> **15和8的最大公约数:1**
> **15和8的最小公倍数:120**
> 【样例说明】
> (1) 本题输出结束后有换行符。
> (2) 输入小于1的数值，提示重新输入，直到满足要求。
> (3) “Input again!”结束后跟换行符。

- 读入数据没有问题
- 我们来看看如何求最大公约数从m,或者n中的任意一个开始到1依次递减取计算，
  $最小公倍数 = m \times n \div 最大公约数$

```c
#include <stdio.h>
int main()
{
    int m, n;
    printf("Input m,n:");
    begin:
    scanf("%d%d", &m, &n);
    if (m<1 || n<1){
        printf("Input again!\n");
        goto begin;
    }

    for (int i = m; i >= 1; i--){
        if (m % i == 0 && n % i == 0){
            printf("%d和%d的最大公约数:%d\n%d和%d的最小公倍数:%d\n", m,
n, i, m, n, m * n / i);
            break;
        }
    }
    return 0;
}
```

- 这里使用了一个不怎么规范的 `goto` 语句，其实这里写一个 `while` 语句也可以完成

```c
#include <stdio.h>
int main()
{
    int m, n;
    printf("Input m,n:");
    scanf("%d%d", &m, &n);
    while(m<1 || n<1){
        printf("Input again!\n");
        scanf("%d%d", &m, &n);
    }

    for (int i = m; i >= 1; i--){
        if (m % i == 0 && n % i == 0){
            printf("%d和%d的最大公约数:%d\n%d和%d的最小公倍数:%d\n", m,
n, i, m, n, m * n / i);
            break;
        }
    }
    return 0;
}
```

#号倒三角

【问题描述】输入行数，利用循环结构输出如下图形。

```
*******
 *****
  ***
   *
```

【样例输入】4

【样例输出】如上图，输出结束有换行符。

```c
#include <stdio.h>
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = n; i > 0; i--){
        for (int j = 1; j <= n - i; j++)
            printf(" ");
        for (int j = 1; j <= 2 * i - 1; j++)
            printf("*");
        printf("\n");
    }
    return 0;
}
```

余数定理

> 【问题描述】中国余数定理：“有物不知几何，三三数余一，五五数余二，七七数余三，问：物有几何？”。编程求1~1000以内所有解。
> 【输入输出样例】
>
> ```
>  52  157  262  367  472
> 577  682  787  892  997
> ```
>
> 【样例输出说明】
> (1) 一行输出5个数，每个数占位5个字符（输出结束后跟换行符）
> (2)该数同时满足：被3除余1，被5除余2，被7除余3

- 这里唯一的问题就是控制输出为统一宽度，我们都知道 `.n%f` 是控制n位小数，而 `%md` 就是输出时占m个字符宽度了，不足会使用空格补全，默认是右对齐，要想要左对齐则 `%-md`

```c
#include <stdio.h>
int main()
{
    int count = 0;
    for (int i = 1; i <= 1000; i++){
        if (i % 3 == 1 && i % 5 == 2 && i % 7 == 3){
            printf("%5d", i);
            count++;
            if (count % 5 == 0)
                printf("\n");
        }
    }
    if (count % 5 != 0)
        printf("\n");
    return 0;
}
```

去除空格和数字

> 【问题描述】读入一串字符串，去除空格和数字字符，输入以回车结束。
> 【样例输入】 `abc 33 de`
> 【样例输出】 `abcde`
> 【样例输出说明】最后输入的回车符不输出。输出结束无换行符。

- 依次一个一个地读取输入的字符串，判断是否满足条件，满足就存下来，不满足就继续读取下一个
- 当然我们可以一边读入一边输出。

```c
#include <stdio.h>
int main()
{
    char a;
    while (scanf("%c", &a) && a != '\n'){
        if ((a < '0' || a > '9') && a != ' ')
            printf("%c", a);
    }
    return 0;
}
```

回文数判断

> 【问题描述】3.2.8 从键盘上输入任意正整数，编程判断该数是否为回文数。所谓回文数就是从左到右读这个数与从右到左读这个数是一样的。例如，12321、4004都是回文数。
> 【样例输入1】 `12321`
> 【样例输出1】 `12321是回文`
> 【样例输入2】 `12`
> 【样例输出2】 `12不是回文`
> 【样例输出说明】本题输出结束后没有换行符。

- 回文数的处理还是更加建议使用字符串进行处理

字符串处理

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
bool isPalindrome(int num)
{
    char str[20];
    sprintf(str, "%d", num);
    int left = 0;
    int right = strlen(str) - 1;
    while (left < right){
        if (str[left] != str[right])
            return false;
        left++;
        right--;
    }
    return true;
}

int main()
{
    int num;
    scanf("%d", &num);
    if (isPalindrome(num))
        printf("%d是回文", num);
    else
        printf("%d不是回文", num);
    return 0;
}
```

纯数字处理

```c
#include <stdio.h>
#include <stdbool.h>
bool isPalindrome(int n)
{
    if (n < 0 || (n % 10 == 0 && n != 0))
        return false;
    int reverseNumber = 0;
    while (n > reverseNumber) {
        int digit = n % 10;
        reverseNumber = reverseNumber * 10 + digit;
        n /= 10;
    }
    return n == reverseNumber || n == reverseNumber / 10;
}

int main()
{
    int n;
    scanf("%d", &n);
    if (isPalindrome(n))
        printf("%d是回文\n", n);
    else
        printf("%d不是回文\n", n);
    return 0;
}
```

下面我们简单地对后面代码做一个逻辑上的解释：

> 首先，如果一个数是负数，或者是以0结尾的正整数（除了0本身），那么它不可能是回文数，因为回文数的最高位不会是0。
> 我们需要将输入的正整数n逆序排列，然后与原数进行比较，以判断是否是回文数。为了实现这一点，我们使用一个变量 `reverseNumber` 来存储逆序排列后的数。
> 在 `while` 循环中，我们不断地从n的最低位开始取出数字，然后加到 `reverseNumber` 的最高位。具体步骤如下：
>
> 1. 每次迭代，先取出n的最低位数字（即 `n % 10` ），记为 `digit` 。
> 2. 将 `reverseNumber` 乘以 `10` ，并加上 `digit` ，这样就把digit添加到 `reverseNumber` 的最高位。
> 3. 将n除以10，相当于将n的最低位去掉，以便下一次迭代。
> 4. 当n小于r `everseNumber` 时，说明已经处理了一半以上的数字。这是因为 `reverseNumber` 是从n的后半部分数字逆序排列而来的。例如，对于输入的数12321，当n变成12时， `reverseNumber` 变成123，已经处理了半以上的数字了。
>
> 最后，我们进行判断：
>
> 1. 如果输入的数n是奇数位数，那么n应该等于 `reverseNumber` 除以10后的值，因为 `reverseNumber` 的最高位是原数的中间位，不需要考虑比较。
> 2. 如果输入的数 `n` 是偶数位数，那么n应该等于 `reverseNumber` ，因为 `reverseNumber` 正好是原数的逆序排列。
> 3. 如果判断条件成立，说明输入的正整数n是回文数；如果判断条件不成立，则不是回文数。
>
> 通过这种方法，我们可以判断一个正整数是否为回文数，而不需要使用字符串。这是因为我们只需要比较数字的各个位数，而不需要考虑字符的顺序。