Assignment_No.2

时间进位

> 【问题描述】编写一个程序，输出当前时间的下一秒。
> 【输入形式】用户在第一行按照“小时:分钟:秒”的格式输入一个时间。
> 【输出形式】程序在下一行输出这个时间的下一秒。
> 【样例输入】 `23:59:59`
> 【样例输出】 `00:00:00`
> 【样例说明】用户按照格式要求输入时间，程序输出此时间的下一秒，输出时每个数字占两位，高位补0。

- 这里我们只需要判断三次，若秒数 `+1` 以后若等于 `60` ，则秒数归零，分数 `+1` 依次类推

```c
#include<stdio.h>
int main()
{
    int h, m, s;
    scanf("%d:%d:%d", &h, &m, &s);
    s++;//这个语句等效为s = s + 1;
    if (s == 60)
    {
        s = 0;
        m++;//这个语句等效为 m = m + 1;
        if (m == 60)
        {
            m = 0;
            h++;//这个语句等效为 h = h + 1;
            if (h == 24)//注意小时的进位条件为24
                h = 0;
        }
    }
    printf("%02d:%02d:%02d", h, m, s);
    return 0;
}
```

时钟指针角度问题

> 【问题描述】普通时钟都有时针和分针。在任意时刻，时针和分针都有一个夹角，并且假设时针和分针都是连续移动的。现已知当前的时刻，试求出在该时刻时针和分针的夹角A（0≤A≤180）。
> 注意：当分针处于0分和59分之间时，时针相对于该小时的起始位置也有一个偏移角度。
> 【输入形式】输入一个24小时制的时间。格式是以英文字符冒号（:）分隔的两个整数m和n，其中m表示时，n表示分。
> 【输出形式】输出一个浮点数A，是时针和分针夹角的角度值。该浮点数保留三位小数。
> 【样例输入】 `8:10`
> 【样例输出】 `175.000`

- 这里我们以12点位置为起点，以分钟为起点计算时针与分钟走过的角度

```c
#include <stdio.h>
#include <math.h>
int main() {
    int m, n;
    scanf("%d:%d", &m, &n);
    // 将时针和分针的时刻转换为角度
    float hour_angle = (m % 12 + (float)n / 60) * 30;
    float minute_angle = n * 6;
    float angle = fabs(hour_angle - minute_angle);// 计算夹角
    angle = fmin(angle, 360 - angle);// 取最小夹角
    printf("%.3f", angle);
    return 0;
}
```

抛球返回路程统计

> 【问题描述】已知一球从高空落下时，每次落地后反弹至原高度的四分之一再落下。编写程序，从键盘输入整数n和m，求该球从n米的高空落下后，第m次落地时经过的全部路程以及第m次落地后反弹的高度，并输出结果。
> 【输入形式】从键盘输入整数n和m，以空格隔开。
> 【输出形式】输出两行：第一行输出总路程，结果保留两位小数。第二行输出第m次落地后反弹的高度，结果保留两位小数。第二行输出结束不换行。
> 【样例输入】 `40 3`
> 【样例输出】 `65.00  0.63`

- 这里我们使用一个 `for` 循环，循环m次就可以了

```c
#include <stdio.h>
int main()
{
    int m;
    float n,sum = 0;
    scanf("%f%d",&n,&m);
    for (int i = 0; i < m; i++){
        sum += n;
        n /= 4.0;
        if(i == m-1) break;
        sum += n;
    }
    printf("%.2f\n",sum);
    printf("%.2f",n);
    return 0;
}
```

统计字符个数

> 【问题描述】输入10个字符，统计其中英文字母、空格、数字字符和其他字符的个数。
> 【输入形式】从键盘输入10个字符。
> 【输出形式】各字符个数。
> 【样例输入】（下划线部分表示输入） `Input 10 characters: Shuer 123?`
> 【样例输出】 `letter=5,blank=1,digit=3,other=1`
> 【样例说明】输入提示符“Input 10 characters: ”中的“:”为英文字符，后加一个空格。输出语句的“=”两边无空格。英文字母区分大小写。必须严格按样例输出格式打印。输出结束不换行。

- 这里是限定了长度的字符串我们可以依次一个一个用 `for` 循环读入以后判断，计数。
- 或者我们一次性读进来以后，再使用 `for` 循环进行判断。

单个字符循环读入

```
#include <stdio.h>
int main() {
    int letters = 0, spaces = 0, digits = 0, others = 0;
    char a;
    printf("Input 10 characters: ");
    for (int i = 0; i < 10; i++) {
        a = getchar();
        if (isalpha(a))
            letters++;
        else if (isspace(a))
            spaces++;
        else if (isdigit(a))
            digits++;
        else
            others++;
    }
    printf("letter=%d,blank=%d,digit=%d,other=%d",
letters,spaces,digits,others);
    return 0;
}
```

整个字符串读入

```
#include <stdio.h>
int main() {
    int letters = 0, spaces = 0, digits = 0, others = 0;
    char a[11];
    gets(a);
    printf("Input 10 characters: ");
    for (int i = 0; i < 10; i++) {
        if (isalpha(a))
            letters++;
        else if (isspace(a))
            spaces++;
        else if (isdigit(a))
            digits++;
        else
            others++;
    }
    printf("letter=%d,blank=%d,digit=%d,other=%d",
letters,spaces,digits,others);
    return 0;
}
```

- 这里读取的时候不能使用 `scanf("%10s",a);`，因为我们的输入是包含有空格的，而 `scanf("%s")` 会在读取到空格时停止。

最大公约数与最小公倍数

> 最大公约数和最小公倍数。
> 【问题描述】
> 输入两个正整数a和b（0≤a,b≤1000000），求其最大公约数和最小公倍数并输出。
> 【输入形式】输入两个整数a和b，以空格分隔
> 【输出形式】输出以空格分隔的两个整数，分别是a、b的最大公约数和最小公倍数。在输出末尾要有一个回车符。
> 【样例输入】`12 18`
> 【样例输出】`6 36`
> 【样例说明】12和18的最大公约数是6，最小公倍数是36。

- 这个题在上几周出现过

```c
#include <stdio.h>
int main()
{
    int m, n;
    scanf("%d%d", &m, &n);

    for (int i = m; i >= 1; i--){
        if (m % i == 0 && n % i == 0){
            printf("%d %d\n", i,  m * n / i);
            break;
        }
    }
    return 0;
}
```

换钱的交易

> 【问题描述】一个百万富翁碰到一个陌生人，陌生人找他谈了一个换钱的计划。该计划如下：我每天给你10万元，而你第一天给我1分钱，第二天我仍给你10万元，你给我2分钱，第三天我仍给你10万元，你给我4分钱。你每天给我的钱是前一天的两倍，直到满n（0≤n≤30）天。百万富翁非常高兴，欣然接受了这个契约。编写一个程序，计算这n天中，陌生人给了富翁多少钱，富翁给了陌生人多少钱。
> 【输入形式】输入天数n（0≤n≤30）
> 【输出形式】分行输出这n天中，陌生人所付出的钱和富翁所付出的钱。输出舍弃小数部分，取整。
> 【样例输入】`30`
> 【样例输出】`3000000    10737418`

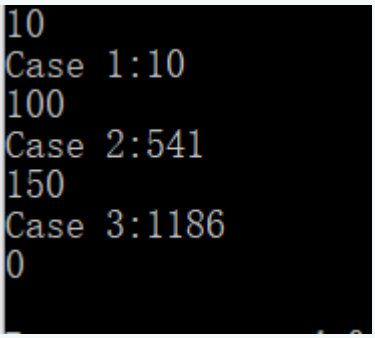
- 题目只需要做一个循环累加就行了

```c
#include <stdio.h>
#include <math.h>
int main() {
    int n;
    scanf("%d", &n);
    long long stranger = 0;
    long long millionaire = 0;

    for (int i = 0; i < n; i++) {
        millionaire += 100000;
        stranger += pow((double)2, (double)i);
    }
    printf("%lld\n%lld", millionaire,stranger/100);//陌生人的计数单位为分
    return 0;
}
```

兑换钱币的游戏

> 【问题描述】对于给定的人民币金额n（单位为元），问有多少种方案将其兑换成1元、2元、5元的组合。
> 【输入形式】输入一个正整数，表示以元为单位的人民币金额n。当输入数值为“0”时结束输入。
> 【输出形式】对于每种情形，先输出“Case #:”（#为序号，从1起），然后依次输出n，逗号，结果，最后换行。



- 这个题目的解法涉及到一个很复杂的算法概念—“动态规划”，但是我们可以使用循环的嵌套来解决这个问题
- 在后面我也会展示一下使用动态规划的算法的解决流程

循环嵌套

```c
#include <stdio.h>
int main() {
    int n, count = 0, Case = 0;

    while(scanf("%d",&n) && n != 0){
        for (int ones = 0; ones <= n; ones++) {
            for (int twos = 0; twos <= n / 2; twos++) {
                for (int fives = 0; fives <= n / 5; fives++) {
                    if ((ones + 2 * twos + 5 * fives) == n) {
                        count++;
                    }
                }
            }
        }
        Case++;
        printf("Case %d:%d\n", Case, count);
    }

    return 0;
}
```

这个算法的逻辑比较简单，它通过三层嵌套循环来尝试所有可能的组合方式，以确定将指定金额n兑换成1元、2元和5元的组合总共有多少种方式。以下是算法的简要逻辑：

1. 使用三个循环变量 `ones` 、 `twos` 和 `fives` 分别代表使用1元、2元和5元的数量。
2. 遍历所有可能的组合方式，从0开始，逐渐增加 `ones` 、 `twos` 和 `fives` 的数量。
3. 在每一轮循环中，计算当前组合的总金额，即 `ones + 2 * twos + 5 * fives` 。
4. 如果当前组合的总金额等于目标金额n，增加计数器count的值，表示找到了一种有效的兑换方式。
5. 继续遍历直到所有可能的组合都被尝试过。
6. 最后， `count` 的值就代表了将金额n兑换成1元、2元和5元的组合的总共方式数量。

尽管这个算法简单易懂，但它在处理较大的n值时会变得非常低效，因为它需要遍历大量的组合，而且嵌套了多层循环。因此，在实际应用中，更常见的做法是使用动态规划等更高效的方法来解决这类组合计数问题。

动态规划

```c
#include <stdio.h>
#include <stdlib.h>

// 计算兑换方案数的函数
int countCombinations(int n) {
    int *dp = (int *)malloc((n + 1) * sizeof(int)); // 使用动态分配的数组来记录兑换方案数
    dp[0] = 1; // 兑换0元的方式只有一种，即不兑换

    // 初始化dp数组
    for (int i = 1; i <= n; i++) {
        dp[i] = 0;
    }

    // 定义硬币面额
    int coins[] = {1, 2, 5};
    int numCoins = sizeof(coins) / sizeof(coins[0]);

    // 遍历硬币面额
    for (int i = 0; i < numCoins; i++) {
        int coin = coins[i];
        for (int j = coin; j <= n; j++) {
            dp[j] += dp[j - coin];
        }
    }

    int combinations = dp[n];
    free(dp); // 释放动态分配的内存

    return combinations;
}

int main() {
    int caseNumber = 1;
    while (1) {
        int n;
        scanf("%d", &n);
        if (n == 0)
            break;
        int combinations = countCombinations(n);
        printf("Case %d:%d\n", caseNumber, combinations);
        caseNumber++;
    }
    return 0;
}
```

1. 我们从0元开始，逐渐计算出凑出每个金额的方法数量。
2. 初始化，我们知道凑出0元的方法只有一种，就是什么都不凑，所以 `dp[0] = 1` 。
3. 现在，我们考虑逐个金额从1元到n元的情况。对于每个金额i，我们要计算凑出它的方法数量。
4. 对于每个金额i，我们有三种可能的方式来凑出它：
    - 使用1元：如果我们用1元来凑，那么问题就变成了凑出(i-1)元的方法数量，所以 `dp[i] += dp[i - 1]` 。
    - 使用2元：如果我们用2元来凑，那么问题就变成了凑出(i-2)元的方法数量，所以 `dp[i] += dp[i - 2]` 。
    - 使用5元：如果我们用5元来凑，那么问题就变成了凑出(i-5)元的方法数量，所以 `dp[i] += dp[i - 5]` 。
5. 重复上述步骤，直到我们计算出了凑出金额n的所有方法数量。
6. 最终， `dp[n]` 就是凑出金额n的不同方法数量。

这种动态规划方法通过逐步计算不同金额的方法数量，然后利用之前计算的结果，有效地找出了将指定金额n兑换成1元、2元和5元的组合方式数量。这样我们就能够了解有多少种不同的方式来凑出给定的金额。

求$\sin x$的近似值

> 【问题描述】
> 已知$\sin x$的近似计算公式如下：
> $$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}$$
> x为弧度，n为正整数。编写程序根据用户输入的x和n，利用上述公式计算$\sin x$的近似值。结果保留8位小数。
> 【输入形式】输入小数 `x (0≤x≤20)` 和整数 `n (1≤n≤5000)` ，两数中间用空格分隔。
> 【输出形式】输出计算结果，保留8位小数。
> 【样例输入1】 `0.5236 4`
> 【样例输出1】 `0.50000105`
> 【样例输入2】 `0.5236 50`
> 【样例输出2】 `0.50000106`

- 这里我们使用到了 `math.h` 里面的 `pow(x,y)` 计算的是x^y的值，注意的是在一部分的编译器里面要求 `pow()` 的两个参数都需要是 `double` 记得强制转换一下

```c
#include <stdio.h>
#include <math.h>
int main() {
    double x, term, result = 0.0;
    int n, sign = 1;
    scanf("%lf%d", &x, &n);
    term = x;
    for (int i = 1; i <= 2*n-1; i += 2) {
        result += (double)sign * term;
        sign = -sign; // 每一项的符号交替变化
        term *= (x * x) / (double)((i + 1) * (i + 2)); // 计算下一项
    }
    printf("%.8lf", result);
    return 0;
}
```

- 这里对这个公式进行了一次优化
    - 我们可以看到$\sin x = \sum_{n}^{i=1} a_n$ ，$a_n = (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}$
    - 进一步我们可以得到递推公式：$a_{n+1} = a_n \times \frac{x^2}{(n+1)\times(n+2)}$

求同构数

【问题描述】设b是a的平方，若a与b的尾部相同，则称a是同构数。例如，25的平方是625，所以25是同构数。编写程序满足如下要求：输入两个整数m和n，找出m、n之间全部的同构数（包括m和n本身）。
【输入形式】输入数据范围的下限m和上限n，要求m和n都为整数，m和n之间用一个空格分隔。
【输出形式】按照由小到大的顺序输出所有同构数，每个整数占一行。若在该范围内没有同构数，则输出字符串 `No Answer` 。
【样例输入1】 `0 30`
【样例输出1】
`0`
`1`
`5`
`6`
`25`
【样例说明1】在0～30之间的同构数有0，1，5，6，25。
【样例输入2】 `100 200`
【样例输出2】 `No Answer`
【样例说明2】在100～200之间，因为没有同构数，所以输出 `No Answer` 。输出结束需要换行。

```c
#include <stdio.h>
#include <stdbool.h>

bool isIsomorphic(int num) {
    long long square = (long long)num * num;
    char str_num[20]; // 假设最大整数为19位
    char str_square[38]; // 对应的平方最多38位

    sprintf(str_num, "%d", num);
    sprintf(str_square, "%lld", square);

    int len_num = 0;
    int len_square = 0;
    while (str_num[len_num] != '\0') {
        len_num++;
    }
    while (str_square[len_square] != '\0') {
        len_square++;
    }

    if (len_square < len_num) {
        return false;
    }

    for (int i = 0; i < len_num; i++) {
```

```
        if (str_num[len_num - 1 - i] != str_square[len_square - 1 -
i]) {
            return false;
        }
    }

    return true;
}

void findIsomorphicNumbers(int m, int n) {
    bool found = false;
    for (int i = m; i <= n; i++) {
        if (isIsomorphic(i)) {
            printf("%d\n", i);
            found = true;
        }
    }

    if (!found) {
        printf("No Answer\n");
    }
}

int main() {
    int m, n;
    scanf("%d %d", &m, &n);
    findIsomorphicNumbers(m, n);
    return 0;
}
```

1. 首先，接收用户输入的范围下限和上限，这两个整数分别用变量 `m` 和 `n` 存储。
2. 创建一个变量 `hasIsomorphic` 用于标记是否在给定范围内找到了同构数，初始值设为0。
3. 使用一个循环，遍历从 `m` 到 `n` 的所有整数。
4. 在循环中，对于每个整数 `num`，计算它的平方并将结果存储在变量 `squared` 中。
5. 然后，逐位比较 `num` 和 `squared` 的每个数字是否相同。可以使用循环或字符串比较来实现此步骤。
6. 如果所有位都相同，将 `num` 输出，并将 `hasIsomorphic` 设置为1，表示已经找到了同构数。
7. 循环结束后，检查 `hasIsomorphic` 的值。如果它仍然为0，表示在给定范围内没有找到同构数，此时打印 `"No Answer"`。

这个算法的核心思想是在指定范围内遍历整数，对每个整数计算其平方，并检查它们的每一位是否相同，以确定是否是同构数。