C5 - Solution

A 计算函数值 2024

难度	考点
1	库函数

题目分析

本题按照公式调用库函数即可，注意 `float` 精度较低，建议大家以后在使用浮点数的时候都使用 `double` 来声明浮点数。

示例代码

```c
#include <stdio.h>
#include <string.h>
#include <math.h>

double f(double x) {
    if(x == 0)
        return 0;
    return sin(x) * log(fabs(x) + 1);
}

int main() {
    double x;
    while(~scanf("%lf", &x)) {
        printf("%.3lf\n", f(x));
    }
    return 0;
}
```

B 小懒獭与函数

难度	考点
2	函数

题目分析

可以实现一个 `f(x)`，然后再输出 `f(f(f(x)))`。

需要注意的是 `x` 的最大值是 10^5，所以 `x * x` 可能超出 `int` 值，注意要用 `long long` 乘法。

`popcount` 可以利用位运算学的知识实现，也可以直接使用库函数 `__builtin_popcount`，感兴趣的同学自己查阅相关库函数。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <math.h>

int popcount(int x) {
    int ret = 0;
    for(int i = 31 ; i >= 0 ;i--) {
        ret += (x >> i) & 1;
    }
    return ret;
}

int f(int x) {
    return (popcount(x) + 1ll * x * x) % 10000;
}

int main() {
    int x;
    while(~scanf("%d", &x)) {
        printf("%d\n", f(f(f(x))));
    }
    return 0;
}
```

C 斐波那契数列 (easy version)

难度	考点
3	递归

题目分析

注意到，这个月新增的小兔数量，等于这个月拥有的大兔子数量。而这个月的大兔子数量，又等于上一个月的兔子总数。

记第一个月的大小兔子数量分别为 a, b，第 n 月的兔子总数为 $f(n)$，那么有如下递推式

$$\begin{cases} a+b & x=1 \\ 2\times a+b & x=2 \\ f(n-1)+f(n-2) & x\geq 3 \end{cases}$$

或者可以写成

$$\begin{cases} a & x=0 \\ a+b & x=1 \\ f(n-1)+f(n-2) & x\geq 2 \end{cases}$$

注意运算中可能存在的爆 `int` 情况，尤其是使用 $2\times a+b$ 的写法，在 $0\leq a,b\leq 10^9$ 的数据范围内，这个表达式是爆 `int` 的。

示例代码

```c
#include <stdio.h>
int a, b;
long long f(int x)
{
    if (x == 0)
        return a;
    else if (x == 1)
        return a + b;
    else
        return f(x - 1) + f(x - 2);
}
int main(void)
{
    scanf("%d%d", &a, &b);
    int t;
    scanf("%d", &t);
    while (t--)
    {
        int n;
        scanf("%d", &n);
        printf("%lld\n", f(n));
    }
    return 0;
}
```

D 卡皮巴拉的素数位置

难度	考点
3	数据类型，素数判断，二进制，函数

题目分析

题目大意：

1. 题目要求判断每只卡皮巴拉所在的位置是否是素数，如果是素数，则根据其位置的序号 i 计算需要准备的玉米数量 2^i 。
2. 素数位置的玉米数量求和后，需判断总玉米数量是否为素数。

解题思路：

1. **判断素数位置**：

   定义一个函数 `is_prime(x)` 来判断 x 是否为素数。我们需要多次使用素数判断，所以可以把素数判断这个部分单独提出来，定义成函数，使得能够复用代码。

   素数范围是 $2\sim10^{10}$ ，所以需要优化，只枚举到 $\sqrt{p}$ ，防止超时(TLE)。

2. **计算玉米总数**：

   遍历卡皮巴拉位置列表，对于每一个素数位置 p_i ，累加 2^i 到总和中。

3. **判断玉米总数是否为素数**：

   再次使用 `is_prime()` 函数判断总和是否为素数。

示例代码

```
#include <stdio.h>

// 判断是否为素数的函数，注意传入参数的数据类型
int is_prime(long long x) {
    // 注意 i 的类型，i * i 可能溢出 int
    // 这里没有特判 2、3，如果 x 是 2 或 3，
    // 因为 i * i <= x 的缘故，不会进入循环
    for (long long i = 2; i * i <= x; i++) {
        if (x % i == 0) return 0;
    }
    return 1;
}

int main() {
    int n;
    // 注意判断最后总玉米数量大小，该用什么数据类型
    long long corn_sum = 0;

    scanf("%d", &n);

    // 遍历每个位置，判断是否为素数并计算玉米数量
    for (int i = 0; i < n; i++) {
        long long p;
        scanf("%lld", &p);
        if (is_prime(p)) {
            printf("Yes\n");
            // 使用左移运算符计算 2^i
            // 因为 i 不重复，可以使用或运算(不进位加法)
            corn_sum |= (1LL << i);
        } else {
            printf("No\n");
        }
    }

    printf("%lld\n", corn_sum);

    if (is_prime(corn_sum)) {
        printf("Yes\n");
    } else {
        printf("No\n");
    }
    return 0;
}
```

拓展补充

本题考查判断**单个素数**。还有一种题型需要求出一个**区间的所有素数**，这种算法我们称为**素数筛法**。感兴趣的同学可以点击链接查看。

E 摩卡与汉诺獭

难度	考点
4	递归，汉诺獭

题目分析

经典递归问题之汉诺塔——课本上就有源代码

让我们再回顾其思路

假设有一个 n 层的汉诺塔在左侧的柱子上，要将其移动到右侧的柱子，需要怎么移动最快?

首先，至少，我们需要将最大的，最底下的第 n 圆盘移动到最右侧的柱子上。

由于圆盘只能放在更大的圆盘上，因此为了将第 n 号圆盘移动到最右侧的柱子上，我们需要：

> 第一步，将前 $n-1$ 个圆盘组成的汉诺塔移动到中间的柱子上
>
> 第二步，将第 n 个圆盘移动到最右侧柱子上
>
> 第三步，将前 $n-1$ 个圆盘组成的汉诺塔移动到右侧柱子上。

第二步可以直接输出，接下来需要处理的就是第一步和第三步，注意到第 n 层圆盘的存在完全不会影响前 $n-1$ 层圆盘的移动，那么第一步和第三步本质就是 $n-1$ 层的汉诺塔的问题。

那么这 $n-1$ 层的汉诺塔问题又可以再一次化归成 $n-2$ 层的汉诺塔，化归成 $n-3$ 层的汉诺塔.....直到变成最基本的情况：只有一个圆盘，直接移动即可。

由此我们就得到了递归关系和初始状态，就能解决整个问题了。

示例代码

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

void move(int n, char from, char to) {
    printf("Moca move otter %d from queue %c to queue %c\n", n, from, to);
}

void hanoi(int n, char from, char via, char to) {
    if(n == 1) {
        move(n, from, to);
        return ;
    }
    hanoi(n - 1, from, to, via);
    move(n, from, to);
    hanoi(n - 1, via, from, to);
}
```

```c
int main() {
    int n;
    scanf("%d", &n);
    hanoi(n, 'A', 'B', 'C');
    return 0;
}
```

F 完美日期

难度	考点
4	判断日期合法性

题目分析

`Hint` 已经提示应该怎样做了，因此我们可以实现相应的函数。

不过由于数据范围太大，如下的暴力代码会运行超时。

```c
#include<stdio.h>
#include<stdbool.h>
#include<math.h>
int l,r,cnt;
bool isdate(int);//判断是否是合法日期
bool isleap(int);//判断是否是闰年。这两个函数的定义与示例代码相同，考虑篇幅限制此处省略。
bool ispow(int);//判断是否是完全平方数
int main(){
    while ( scanf("%d%d",&l,&r)!=EOF ){
        cnt=0;
        for ( int i=l ; i<=r ; i++ )
            if ( isdate(i) && ispow(i) )
                cnt++;
        printf("%d\n",cnt);
    }
    return 0;
}
bool ispow(int x){
    int c=sqrt(x);
    return c*c==x;
}
```

那么我们可以进行适当的优化，比如循环遍历两个端点的算术平方根之间的所有数值，并判断它们的平方是不是合法的日期表示。

示例代码

```c
#include<stdio.h>
#include<stdbool.h>
#include<math.h>
int l,r,cnt;
```

```
bool isdate(int);//判断是否是合法日期
bool isleap(int);//判断是否是闰年。
int main(){
    while ( scanf("%d%d",&l,&r)!=EOF ){
        cnt=0;
        for ( l=sqrt(l-1)+1,r=sqrt(r) ; l<=r ; l++ )//对 l,r 开根，其中对 l 的赋值相当于向上取整
            if ( isdate(l*l) )
                cnt++;
        printf("%d\n",cnt);
    }
    return 0;
}
bool isdate(int x){
    int y=x%10000,m=x/100%100,d=x%100;
    switch ( m ){
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            return d>=1&&d<=31;
        case 4:
        case 6:
        case 9:
        case 11:
            return d>=1&&d<=30;
        case 2:
            return d>=1&&(isleap(y)?d<=29:d<=28);
        default:
            return false;
    }
}
bool isleap(int y){
    if ( y%4==0 && y%100!=0 )
        return true;
    else if ( y%400==0 )
        return true;
    return false;
}
```

Gshtog的电子日历

难度	考点
4	循环结构, 分支结构, 日期计算, 格式化输出

题目分析

这道题就是输入一个年份和月份，然后打印出这个月的月历表。那么首先显然我们需要知道当月第一天是周几，还需要知道当月一共有多少天，才能确定这个月历表的内容。

如何确定当月第一天是周几，可以用上课PPT上的`zeller`公式直接计算第一天是周几，也可以循环推算，使用`zeller`公式的方式`C4`上机出现过，这里讲一下循环推算周几的思路。首先根据提示可以知道1900年1月1日是周一，那么可以循环到输入日期，每过一年就加上这一年的天数，最后加上输入年份的前几个月的天数，算出来从19000101到输入月的总天数，然后对7取模就可以算出输入日期第一天是周几。

如何确定当月一共有多少天，这个首先要判断当年是否是闰年，再判断是否是2月，如果不是2月，那么判断是否是`1 3 5 7 8 10 12`这几个31天月，不是则是30天的月。

最后注意按照格式输出，包括：左右对齐、换行、空格数量等要求，详见代码。

示例代码

```c
#include <stdio.h>

int main() {
    int y, m;
    scanf("%d%d", &y, &m);
    int sumDays = 0;
    int i;
    for (i = 1900; i < y; ++i) {
        // 判断i是否是闰年
        int isleap = ((i % 4 == 0 && i % 100 != 0) || (i % 400 == 0));
        sumDays += isleap ? 366 : 365;
    }
    int isleap = (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);
    int table[20] = {0, 31, 28 + isleap, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    for (i = 1; i < m; ++i) sumDays += table[i]; // 算出到目标月的总天数来推算目标月第一天是周几
    sumDays++; // 加上当月第一天
    int mDays = table[m];
    int dayOne = sumDays % 7; // 首行第一个数字在第dayOne个位置(从0起)
    printf("Sun Mon Tue Wed Thu Fri Sat\n");
    // 判断1号在第一行的位置
    for (i = 1; i <= mDays + dayOne; ++i) {
        if (i <= dayOne) printf("    "); // 第一行刚开始，先输出空格
        else printf("%3d", i - dayOne); // %3d表示输出要占3个位置+右对齐
        if (i == mDays + dayOne) break; // 最后一次循环及时终止，不再继续输出了
        if (i % 7 == 0) printf("\n");
        else printf(" ");
    }
    return 0;
}
```

H 九连环

难度	考点
5	函数的调用、递归、位运算

题目分析

Hint 给出了步数最少的解法，由于篇幅有限故不作最优解证明。

我们只需实现两个函数，分别表示放上 k 连环和取下 k 连环即可（Hint 给出的流程保证了取下 k 连环时所有环都在架上，放上 k 连环时所有环都在架下，因此只需递归调用即可）。不要忘记每次操作时更新 $status$ 。由于两个函数互相调用，因此我们应当在函数定义前完成两个函数的声明。

示例代码

```
#include<stdio.h>
int n,status;
void up(int);//放上 k 连环
void down(int);//取下 k 连环
int main(){
    scanf("%d",&n);
    status=(1<<n)-1;//初始化 status
    down(n);
    return 0;
}
void up(int k){//与 down(k) 完全反向即可
    if ( k>0 ){
        up(k-1);
        down(k-2);
        printf("%d\n",status^=1<<k-1);
        up(k-2);
    }
}
void down(int k){
    if ( k>0 ){
        down(k-2);
        printf("%d\n",status^=1<<k-1);
        up(k-2);
        down(k-1);
    }
}
```

I 摩卡与艺术家水獭

难度	考点
6	递归，函数，记忆化，动态规划

题目分析

通过观察可得：

- 如果当前高度为 n，且不是最后一层，则下面至少有一黑一白，可以将问题分解为高度为 $n-2$ 的结果 $S(n-2)$ 和高度为 $n-(k+1)$ 的结果 $S(n-(k+1))$（即最上面黑色白色的搭配要么是 `1 - 1`，要么是 `k - 1`）。
- 如果当前高度是 n，且是最后一层（单独一个黑色），那么当剩余高度 `i` 满足 `i >= 1` 时，可以构成一层，`ret += 1`；当剩余高度 `i` 满足 `i >= k` 时，可以构成 `k` 层，`ret += 1`。

递归关系可以表示为（`tem` 表示 $n \geq 1$ 和 $n \geq k$ 两式中成立式子的个数）：

$$S(n) = S(n-2) + S(n-(k+1)) + tem$$

但是直接用这个表达式写一个递归函数会超时，效率低在哪里了呢？当我们计算比如 $S(20)$ 时，不管是第几次调用 $S(20)$，他总是会重新计算 $S(18)$ 和 $S(19-k)$ 的值，其实这些值我们之前也就算过了。我们想，我们既然算过了这些值，就可以把它们存起来，等到下一次调用的时候直接读取存下来的结果，这就是 **记忆化** 的思想。存储的功能，可以用数组实现。

示例代码

```
#include <stdio.h>

int l, k;
long long ans;
long long dp[105];

// 表示剩余的高度和上一层的合法数目
long long dfs(int i) {
    if(i <= 0)
        return 0;

    // 记忆化：算过一次就不重新计算
    if(dp[i])
        return dp[i];

    int tem = 0;
    if(i >= 1)
        tem += 1;
    if(i >= k)
        tem += 1;
    return dp[i] = dfs(i - k - 1) + dfs(i - 2) + tem;
}

int main() {
    scanf("%d%d", &l, &k);
    dp[1] = 1;
    printf("%lld", dfs(l));
    return 0;
}
```