第二次编程练习报告

姓名：张洋　　学号：2111460　　班级：信安二班

编程练习 1——编程实现平方-乘算法

```
#include<iostream>
#include<sstream>
using namespace std;

void dec_to_bin(string& s, int n) {//n 十进制转二进制
    while (n > 0) {
        s.insert(0, to_string(n % 2));
        n /= 2;
    }
}
int square_and_multiply(int a, int m, string s) {//平方-乘算法
    long long int c = 1;
    for (int i = 0; i < strlen(s.c_str()); i++) {
        c = c * c % m;
        if (s[i] == '1')
            c = c * a % m;
    }
    return c;
}
int main() {
    int a, n, m;
    cout << "Calculate a^n(mod m)..." << endl;
    cout << "Please input:" << endl;
    cout << "  a=";
    cin >> a;
    cout << "  n=";
    cin >> n;
    cout << "  m=";
    cin >> m;
    string s;//十进制转二进制后存入字符串
    dec_to_bin(s, n);
    cout << a << "^" << n << "(mod " << m << ")=" << square_and_multiply(a, m, s) << endl;
    system("PAUSE");
}
```

说明部分：

算法思路如下：要计算 $a^m \pmod n$，设 m 的二进制表示为：

$$
\begin{aligned}
m &= m_{k-1}2^{k-1} + m_{k-2}2^{k-2} + \cdots + m_1 2^1 + m_0 \\
&= 2(2(\cdots(2(2m_{k-1} + m_{k-2}) + m_{k-3})\cdots) + m_1) + m_0,
\end{aligned}
$$

于是有

$$
\begin{aligned}
a^m &\equiv a^{m_{k-1}2^{k-1}+m_{k-2}2^{k-2}+\cdots+m_1 2^1+m_0} \pmod n \\
&\equiv ((\cdots((a^{m_{k-1}})^2 a^{m_{k-2}})^2 \cdots a^{m_2})^2 a^{m_1})^2 a^{m_0} \pmod n.
\end{aligned}
$$

根据这一表达式，可以设计计算模幂的快速算法，算法过程如下：

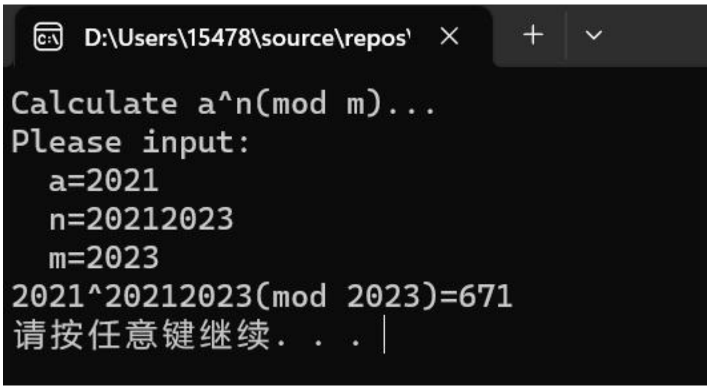
算法 2.3.1　平方-乘算法

输入： a，幂次m，模n;

输出： $a^m \pmod n$的结果c;

1. $c \leftarrow 1$;
2. **FOR** $i = k-1$ **TO** 0
3. 　　$c \leftarrow c^2 \pmod n$;
4. 　　**IF** $m_i = 1$ **THEN**
5. 　　　　$c \leftarrow c \cdot a \pmod n$
6. 　　**END IF**
7. **RETURN** c;

运行示例：



编程练习 2——编程实现扩展的欧几里得算法求逆元

```
#include<iostream>
using namespace std;
void swap(int& a, int& b) {
    int t = a;
    a = b;
    b = t;
}
```

```
int inverse(int x, int y) { //求x模y的乘法逆元
    int flag = 0; //记录是否交换x, y
    if (x < y) { //令x为较大的那一个
        flag = 1;
        swap(x, y);
    }
    //以下利用扩展欧几里得算法求乘法逆元
    int i = 1, s[100], t[100], r[100], q[100];
    r[0] = x;
    r[1] = y;
    s[0] = t[1] = 1;
    s[1] = t[0] = 0;
    while (r[i] != 0) { //当余数不为0时
        q[i] = r[i - 1] / r[i];
        s[i + 1] = s[i - 1] - q[i] * s[i];
        t[i + 1] = t[i - 1] - q[i] * t[i];
        i++;
        r[i] = r[i - 2] % r[i - 1];
    }
    if (flag == 1) {
        if (t[i - 1] < 0)
            return t[i - 1] + x;
        else
            return t[i - 1];
    }
    else {
        if (s[i - 1] < 0)
            return s[i - 1] + y;
        else
            return s[i - 1];
    }
}
int gcd(int a, int b) { //利用辗转相除法求最大公因数
    if (a < b) //a为较小的那个数
        swap(a, b); //交换ab,让a为较大的数
    int r = a % b; //余数
    while (r != 0) { //当余数不为0时
        a = b;
        b = r;
        r = a % b;
    }
    return b;
}
int lcm(int a, int b) { //最小公倍数
```

```
    return a * b / gcd(a, b);
}
int main() {
    int a, b;
    cout << "a=";
    cin >> a;
    cout << "b=";
    cin >> b;
    cout << "gcd(a,b)=" << gcd(a, b) << endl;
    cout << "lcm(a,b)=" << lcm(a, b) << endl;
    cout << "a^(-1)=" << inverse(a, b) << "(mod " << b << ")" << endl;
    cout << "b^(-1)=" << inverse(b, a) << "(mod " << a << ")" << endl;
    system("PAUSE");
}
```

说明部分：

设 r_0, r_1是两个正整数，且$r_0 > r_1$，设$r_i(i = 1,2, \ldots\ldots, n)$是使用欧几里德算法计算$(r_0, r_1)$时所得到的余数序列且$r_{n+1} = 0$，则可以使用如下算法求整数$s_n$和$t_n$，使得

$$(r_0, r_1) = s_n r_0 + t_n r_1.$$

这里s_n和t_n是如下递归定义的序列的第 n 项，且

$$s_0 = 1,\ t_0 = 0;$$

$$s_1 = 0,\quad t_1 = 1;$$

$$s_i = s_{i-2} - q_{i-1}s_{i-1},\quad t_i = t_{i-2} - q_{i-1}t_{i-1},\ \text{其中}\ q_i = r_{i-1}/r_i,\ i = 2, 3, \ldots, n.$$

运行示例：