# 第二次编程练习报告

姓名：周钰宸 学号：2111408 班级：信安一班

##### **编程练习1——编程实现平方-乘算法**

* **源码部分：**

#include<iostream>

#include<stack>

using namespace std;

stack<int> DecimalToBinary(unsigned long long n) {//计算十进制转二进制，用一个栈s来保存

stack<int> s;

while (n!= 0) {

int r = n % 2;

s.push(r);

n = n / 2;

}

return s;

}

unsigned long long SquareAndMultiplyAlgorithm(unsigned long long a, unsigned long long n, unsigned long long m) {

unsigned long long result = 1;//result定义为64位的整型,防止中间结果超过32位而发生溢出

stack<int> s = DecimalToBinary(n);

while (!s.empty()) {//由于算法的特性，先要处理的是二进制的头位，即在二进制转为十进制方法中最后得到的余数

//符合FILO的特性，故采用栈的形式存储并操作

result = (result\*result) % m;

if(s.top()==1){

result = (result \*a) % m;

}

s.pop();

}

return result;

}

int main() {

cout << "Calculate a^n(mod m)..." << endl;

cout << "Please inpust:" << endl;

unsigned long long a;

cout << " a=";

cin >> a;

unsigned long long n;

cout << " n=";

cin >> n;

unsigned long long m;

cout << " m=";

cin >> m;

cout << a << "^" << n << "(mod " << m << ")=" << SquareAndMultiplyAlgorithm(a,n,m)<< endl;

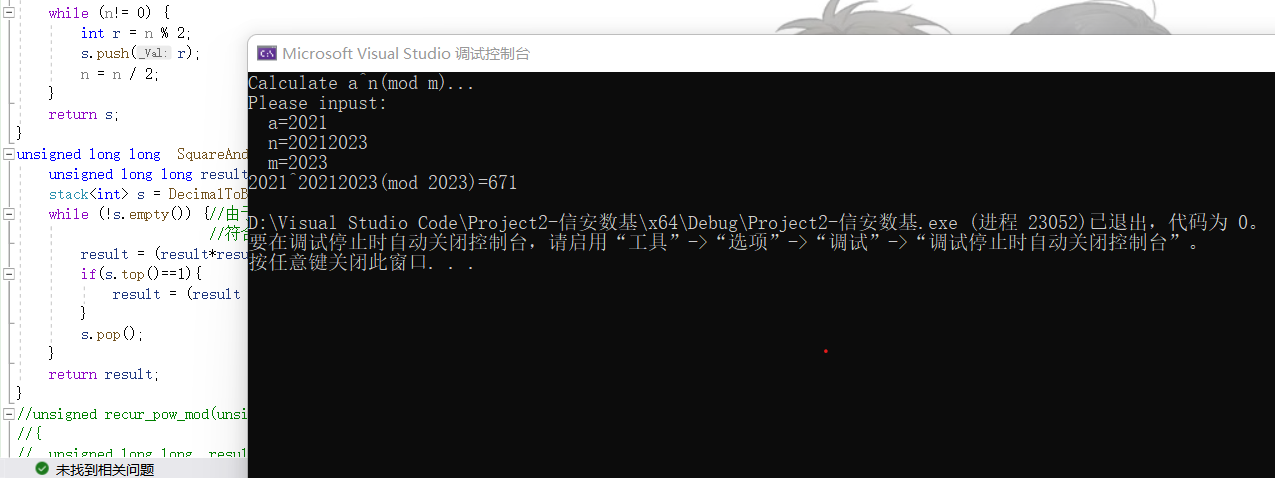
return 0;

}

* **说明部分：**

1. 先编写一个函数DecimalToBinary用来将十进制转为二进制，并且利用一个栈去保存。
2. 在SquareAndMultiplyAlgorithm函数中，将返回值和参数类型都设置为unsigned long long类型，并且定义一个unsigned long long的result保存最终结果。
3. 之所以要用一个栈去保存，是因为我注意到在用辗转相除将十进制转为二进制时，先获得的余数是二进制的末尾，它正好在平方乘算法里面是需要最后操作的，因此符合FILO的特性，我采用一个栈去存储二进制结果。然后在SquareAndMultiplyAlgorithm函数中，只要栈不为空，就依次取出栈顶元素（即先处理二进制的靠前位）
4. 最后输出平方乘算法计算得到的高次模幂运算结果。

* **运行示例：**



* **其他：思考问题——**

1. 十进制转为二进制的另一种操作方式：

答：采用位运算进行操作——

unsigned recur\_pow\_mod(unsigned a, unsigned b, unsigned mod)

{

unsigned long long result = 1; //这里result定义为64位的整型是因为有时result的中间值会超过32位，如果只用unsigned会溢出

unsigned array[64] = { 0 }; //用来存储b的二进制形式的每一位数

unsigned length = 0;

while (b)

{

array[length++] = b & 1;

b >>= 1;

}

for (int j = length - 1; j >= 0; j--)

{

if (array[j] == 1)

result = (result \* result \* a) % mod;

else

result = (result \* result) % mod;

}

return result;

}

##### **编程练习2——编程实现扩展的欧几里得算法求逆元**

* **源码部分：**

#include<iostream>

#include<vector>

using namespace std;

void ExtendedEuclideanAlgorithm(int result[],int a,int b) {//扩展的欧几里得算法

//result数组，result[0]存gcd最大公因数，result[1]存最小公倍数

//result[2]存a^(-1)(mod b),result[3]存b^(-1)(mod a)

int r0, r1;

a > b ? r0 = a, r1 = b : r0 = b, r1 = a;//先判断较大的作为r0，较小的作为r1

vector<int> v\_r, v\_q, v\_s, v\_t;//分别用来存放ri,qi,si,ti的vector类型变量

v\_r.push\_back(r0), v\_r.push\_back(r1);

v\_q.push\_back(-1);//q0用不上，附上特殊值

v\_s.push\_back(1), v\_s.push\_back(0);//s0=1,s1=0

v\_t.push\_back(0), v\_t.push\_back(1);//t0=0,t1=1

//在进行计算之前，先初始化操作

while (v\_r[v\_r.size()-2]%v\_r[v\_r.size()-1]){//只要余数不为0

v\_q.push\_back(v\_r[v\_r.size() - 2] / v\_r[v\_r.size() - 1]);

v\_r.push\_back(v\_r[v\_r.size() - 2] % v\_r[v\_r.size() - 1]);

v\_s.push\_back(v\_s[v\_s.size() - 2] - v\_q[v\_q.size()-1] \* v\_s[v\_s.size() - 1]);

v\_t.push\_back(v\_t[v\_t.size() - 2] - v\_q[v\_q.size()-1] \* v\_t[v\_t.size() - 1]);

}

result[0] = v\_r[v\_r.size() - 1], result[1] = a \* b / result[0];

if (a > b) {//r0=a,r1=b

if (v\_s[v\_s.size() - 1] < 0) {//判断一下系数是否为负，进行特殊处理

result[2] = b + v\_s[v\_s.size() - 1];//result[2]存a^(-1)(mod b),用b+

}

else {

result[2] = v\_s[v\_s.size() - 1];

}

if (v\_t[v\_t.size() - 1] < 0) {

result[3] = a + v\_t[v\_t.size() - 1];//result[3]存b^(-1)(mod a)，用a+

}

else {

result[3] = v\_t[v\_t.size() - 1];

}

}

else {//r0=b,r1=a

if (v\_s[v\_s.size() - 1] < 0) {//判断一下系数是否为负，进行特殊处理

result[3] = a + v\_s[v\_s.size() - 1];//result[3]存b^(-1)(mod a)，用a+

}

else {

result[3] = v\_s[v\_s.size() - 1];

}

if (v\_t[v\_t.size() - 1] < 0) {

result[2] = b + v\_t[v\_t.size() - 1];//result[2]存a^(-1)(mod b),用b+

}

else {

result[2] = v\_t[v\_t.size() - 1];

}

}

return;

}

int main() {

int a;

cout << "a=";

cin >> a;

int b;

cout << "b=";

cin >> b;

int result[4];

for (int i = 0; i < 4; i++) {

result[i] = 0;

}

ExtendedEuclideanAlgorithm(result, a, b);

cout << "gcd(a,b)=" << result[0] << endl;

cout << "lcm(a,b)=" << result[1] << endl;

cout << "a^(-1)=" << result[2] << "(mod " << b << ")" << endl;

cout << "b^(-1)=" << result[3] << "(mod " << a << ")" << endl;

return 0;

}

* **说明部分：**

1. 定义ExtendedEuclideanAlgorith即扩展欧几里得算法实现求逆元。采用4个vector类型的数组分别保存ri,qi,si,ti，先添加一些初始元素进行初始化。然后只要余数不为0，就按照算法实现的公式在重复进行在vector后添加下一步操作后的元素，当余数位0时跳出循环。
2. 采用一个result数组，result[0]保存最大公因数，result[1]保存最小公倍数，result[2]和result[3]保存两个结果的逆元。
3. 最后在main函数中测试和输出。

* **运行示例：**

