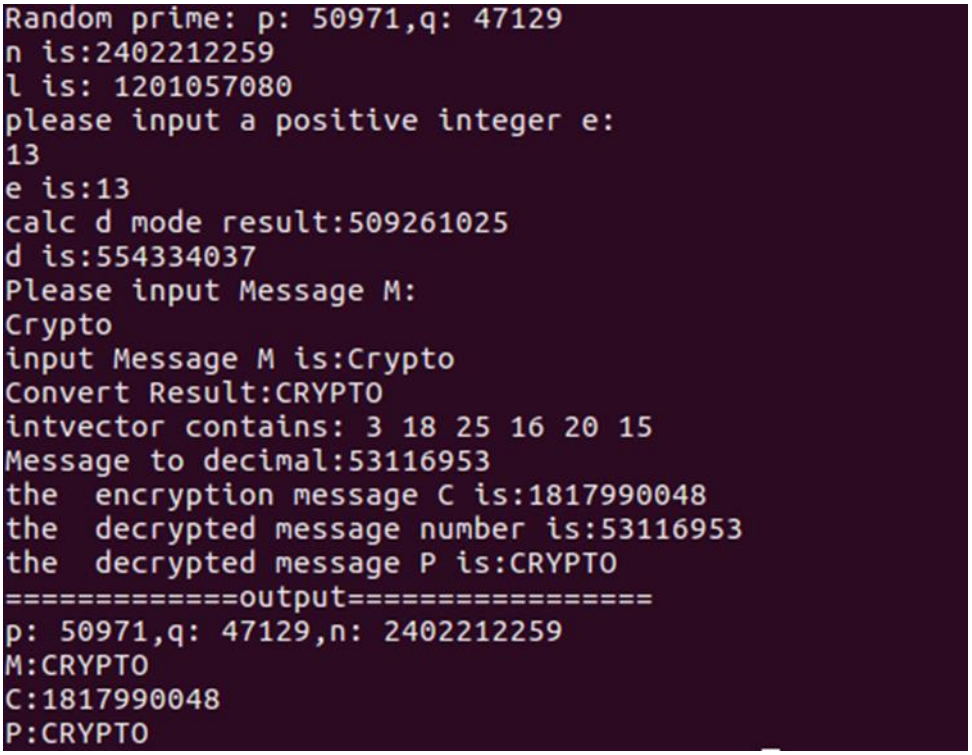
Advanced Algorithms: RSA Assignment

Anshul Gautam

Program output



Program code

```
#include <iostream>
#include<ctime>
#include <stdio.h>
#include<cmath>
#include<stdlib.h>
#include <vector>


//Algorithm for RSA encryption and decryption


using namespace std;

unsigned long Get_Miller_Rabin_PrimeNum(unsigned long);
unsigned long SteinGcd(unsigned long , unsigned long );
unsigned long EuclidGcd(unsigned long , unsigned long );
```

```
unsigned long Euclid(unsigned long , unsigned long );
unsigned long PowMod(unsigned long , unsigned long , unsigned long );
unsigned long StrEncode(char * );
vector<char>  strDecode(unsigned long );
unsigned long  lcm(int p,int q);
unsigned long RandomPrime(char bits);


// StrEncode: This function takes in characters as input and converts them to the base 27.
// ASCII values of characters have been used to perform the conversion.
unsigned long StrEncode(char * input) {

	// cout<<"RESULT:"<<input<<endl;
	// printf("RESULT:%s",&input);
	unsigned long returnValue=0;
	char *p = input;
	vector<int> intArray(5);
	vector<int>::iterator it;
	int idx=0;
	int tempvalue=0;
	it=intArray.begin();
	intArray.insert(intArray.begin(),100);

	//cout<<"RESULT:"<<p<<endl;
	if(p==NULL) {

		cout<<"input str is null,error"<<endl;
		return 0;

	}

	while(*p) {
		//cout<<"input char:"<<*p<<endl;
		if ('A' <= *p && *p <= 'Z') {
			tempvalue=(int)(*p-64);
			intArray[idx]=tempvalue;
			//std::cout << "temporary value" << tempvalue << endl;
			//intArray.insert(it,tempvalue);
		} else if ('a' <= *p && *p <= 'z') {
			//cout<<"input char:"<<*p<<endl;
			*p = *p-32;
			tempvalue=(int)(*p-64);
			intArray[idx]=tempvalue;
			//std::cout << "temporary value " << tempvalue << endl;
			//intArray.insert(it,tempvalue);
			//intArray.insert(it,64);
			//cout<<"input char after:"<<*p<<endl;
		} else if(*p==' ') {
			//intArray.insert(it,0);
```

```
                intArray[idx]=0;
            }
            p++;
            idx++;
        }

        cout<<"Convert Result:"<<input<<endl;


        // display values
        std::cout << "intvector contains:";
        for (it=intArray.begin(); it<intArray.begin()+idx; it++)
            std::cout << ' ' << *it;
        std::cout << '\n';



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

            //cout << "power " << pow(27, idx-i-1) << endl;
            returnValue+=intArray[i]*pow(27,idx-1-i);


        }


        return returnValue;
}

// Decode the input to find the characters used while encoding the message.
vector<char> strDecode(unsigned long input) {

        vector<char> result;
        vector<char>::iterator it;
        it=result.begin();
        char charTemp;
        int size=0;
        int modResult=0;
        int divisionResult=0;
        do {

            divisionResult=input/27;
            modResult=input%27;
            //cout<<"Convert char int  Result:"<<modResult<<endl;
            charTemp=(char)(modResult+64);
            result.push_back(charTemp);
            //cout<<"Convert char Result:"<<charTemp<<endl;
          input=divisionResult;
          size++;
```

```
            it++;

        } while(divisionResult>0);

        // display values
        /*
        std::cout << "intvector contains:";
        for (it=result.begin(); it<result.begin()+size; it++)
                std::cout << ' ' << *it;
        std::cout << '\n';
*/



        vector<char> real_result;
        for(int i=size-1;i>=0;i--) {

                real_result.push_back(result[i]);
        }
        // display values


        for (it=real_result.begin(); it<real_result.begin()+size; it++)
                std::cout << *it;
        std::cout << '\n';

  return real_result;
}

const static long       g_PrimeTable[]=
{
   3,5,7,11,13,17,19,23,29,31,37,41,43,
   47,53,59,61,67,71,73,79,83,89,97
};

const static long       g_PrimeCount=sizeof(g_PrimeTable) / sizeof(long);
const unsigned long  multiplier=12747293821;
const unsigned long  adder=1343545677842234541;// Random class
class                RandNumber
{
private:
   unsigned long    randSeed;
public:
   RandNumber(unsigned long s=0);
   unsigned long    Random(unsigned long n);
};
RandNumber::RandNumber(unsigned long s)
{
   if(!s)
```

```
   {
      randSeed= (unsigned long)time(NULL);
   }
   else
   {
      randSeed=s;
   }
}

unsigned long RandNumber::Random(unsigned long n)
{
   randSeed=multiplier * randSeed + adder;
   return randSeed % n;
}static RandNumber   g_Rnd;


/*
Calculate the greatest common divisor using Euclid Algorithm
*/
unsigned long EuclidGcd(unsigned long p, unsigned long q) {
        unsigned long   a=p > q ? p : q;
        unsigned long   b=p < q ? p : q;
        unsigned long   t;
        if(p == q) {
                return p;   //Two numbers are equal, the greatest common divisor is itself
        } else {
                while(b) {  // Euclidean algorithm , gcd(a,b)=gcd(b,a-qb)
                        a=a % b;
                        t=a;
                        a=b;
                        b=t;
                }
                return a;
        }
}


/*
Calculate the greatest common divisor using Stein
*/
unsigned long SteinGcd(unsigned long p, unsigned long q) {
        unsigned long   a=p > q ? p : q;
        unsigned long   b=p < q ? p : q;
        unsigned long   t, r=1;

        if(p == q) {
                return p;          //Two numbers are equal, the greatest common divisor is itself
        } else {
                while((!(a & 1)) && (!(b & 1))) {
```

```
                r<<=1;          //when a and b are even, gcd(a,b)=2*gcd(a/2,b/2)
                a>>=1;
                b>>=1;
            }
            if(!(a & 1)) {
                t=a;            //a is even, exchange a and b
                a=b;
                b=t;
            }
            do {
                while(!(b & 1)) {
                    b>>=1;      //b is even, a is odd, then gcd(b,a)=gcd(b/2,a)
                }
                if(b < a) {
                    t=a;        //b is less than a, exchange a and b
                    a=b;
                    b=t;
                }
                b=(b - a) >> 1; //a and b are both odd, gcd(b,a)=gcd((b-a)/2,a)
            } while(b);
            return r * a;
        }
}


// Modular multiplication, Return x=a*b mod n
//（a*b）%p=（a%p*b%p）%p
inline unsigned long MulMod(unsigned long a, unsigned long b, unsigned long n) {

        //test overflow
    //cout<<"a:"<<a<<"\t b:"<<b<<endl;
        unsigned long long temp1=(unsigned long long)a*(unsigned long long)b;
        if((temp1/a)!=b)
        {
            cout<<"temp1:"<<temp1<<endl;
            cout<<"a:"<<a<<"\t b:"<<b<<endl;
            printf("multiplication overflow!!!!!\n");
        }

        /*
        a=a%n;
        b=b%n;
        cout<<"a:"<<a<<"\t b:"<<b<<endl;

        temp1=a*b;
        if((temp1/a)!=b)
        {
            cout<<"a:"<<a<<"\t b:"<<b<<endl;
            printf("Still multiplication overflow!!!!!\n");
```

```
	}
	*/
	unsigned long temp4=temp1%n;
	return temp4;
	//return a * b % n;
}
//Modular multiplication, return x=base^pow mod n
unsigned long PowMod(unsigned long base, unsigned long pow, unsigned long n) {
	long long	a=base, b=pow, c=1;

	/*

		long long temp=1;
	   for(int i=0;i<pow;i++)
	   {
		temp=temp*base;
		}
	   return temp%n;

	   */

	while(b) {
		while(!(b & 1)) {
			b>>=1;		//a=a * a % n;	// It seems the function can deal with a 64-bit
integer, since here a*a cause an overflow error when a>=2^32, so the actual processing range cannot
achieve 64-bit
			a=MulMod(a, a, n);
		}
		b--;		//c=a * c % n;	// Also cause an overflow, maybe split 64-bit into 32-bit
		c=MulMod(a, c, n);
	}
	return c;


}
/*
Known a and b, find x that satisfies a*x =1 (mod b)
Which is equivalent to find minimal integral solution of a*x-b*y=1
*/
unsigned long Euclid(unsigned long a, unsigned long b) {
	unsigned long	m, e, i, j, x, y;
	unsigned long	 xx, yy;
	m=b;
	e=a;
	x=0;
	y=1;
	xx=1;
	yy=1;
	while(e) {
```

```
                i=m / e;
                j=m % e;
                m=e;
                e=j;
                j=y;
                y*=i;
                if(xx == yy) {
                        if(x > y)
                                y=x - y;
                        else {
                                y-=x;
                                yy=0;
                        }
                } else {
                        y+=x;
                        xx=1 - xx;
                        yy=1 - yy;
                }
                x=j;
        }
        if(xx == 0)
                x=b - x;
        return x;
}

// Find the LCM of p and q
unsigned long  lcm(int p,int q) {
        unsigned long  a=1,b=1,all=1;
        if(q>0&&p>0)
        {
                if(p>q)         {
                        a=q;
                } else          {
                        a=p;
                }
                for(b=a; b>=2; b--)         {
                        if(p%b==0&&q%b==0)              {
                                p=p/b;
                                q=q/b;
                                all=all*b;
                        }
                }
                all=all*p*q;
                return all;
        } else    {
                printf("ERROR in lcm!\n");
                return 0;
        }
}
```

```
/*
Rabin-Miller prime number test,   return 1 when passing test, return 0 otherwise
n is the number to test
*/
long RabinMillerKnl(unsigned long n)
{
   unsigned long    b, m, j, v, i;
   m=n - 1;
   j=0;    // Work out m and j which let n-1=m*2^j, m is positive odd, j is nonnegative integer
   while(!(m & 1))
   {
      ++j;
      m>>=1;
   }    // chose a random number b, 2<=b<n-1
   b=2 + ((unsigned)time(NULL)*multiplier+adder)%(n-3);    //calculate v=b^m mod n
   v=PowMod(b, m, n);    // If v==1, passing test
   if(v == 1)
   {
      return 1;
   }    //let i=1
   i=1;    //v=n-1, test passed
   while(v != n - 1)
   {
      //i==l, no prime number, end
      if(i == j)
      {
         return 0;
      }       //v=v^2 mod n,   i=i+1
      unsigned long long xxx;
      int xxxx = 2;
      xxx = xxxx;
      v = PowMod(v, xxx, n);
      ++i;       //循环到5
   }    return 1;
}
/*
Rabin-Miller prime number test, recursive call
Return 1 if all passed, 0 otherwise
*/
long RabinMiller(unsigned long n, long loop)
{
   // little r=prime number building method, improve efficiency
   for(long i=0; i < g_PrimeCount; i++)
   {
      if(n % g_PrimeTable[i] == 0)
      {
         return 0;
```

```
        }
    }    // Recursive call Rabin-Miller tests loop times, let the probability that no prime number can pass the test under than (1/4)^loop
    for(long i=0; i < loop; i++)
    {
        if(!RabinMillerKnl(n))
        {
            return 0;
        }
    }    return 1;
}

/*
Generate an binary prime number randomly, at most 32-bit .
*/
unsigned long RandomPrime(char bits)
{
    unsigned long    base;
    do
    {
        base= (unsigned long)1 << (bits - 1);   // ensure the top digit=1
        base+=g_Rnd.Random(base);               // Add a random number
        base|=1;    // ensure the least digit=1, that ensure odd number
    } while(!RabinMiller(base, 30));    //30 times of Rabin Miller test
    return base;    // it's prime number if all passed
}


// main: Encrypt and Decrypt the messages.
int main(int argc, char** argv) {
        srand((unsigned)time(NULL));
        unsigned long e;//=5;
        unsigned long p;//=17;
        unsigned long q;//=19;
        p=RandomPrime(16);
        q=RandomPrime(16);
        printf("Random prime: p: %lu,q: %lu \n",p,q);
        unsigned long n=p*q;
        cout<<"n is:"<<n<<endl;
        unsigned long l = lcm(p-1,q-1);//(p - 1) * (q - 1);
        printf("l is: %lu\n",l);
        do {
                cout<<"please input a positive integer e:"<<endl;
                cin>>e;
        } while(e>=l||SteinGcd(e,l)!=1);

        cout<<"e is:"<<e<<endl;
        unsigned long d;
        d=Euclid(e,l);
```

```
        //check d
        unsigned long long temp=e*d;
        cout<<"calc d mode result:"<<(temp%l)<<endl;

        cout<<"d is:"<<d<<endl;
        cout<<"Please input Message M:"<<endl;
        char M[10];
        scanf("%s",&M);
        cout<<"input Message M is:"<<M<<endl;
        unsigned long m_long=StrEncode(M);
        cout<<"Message to decimal:"<<m_long<<endl;
        unsigned long C=PowMod(m_long, e, n);
        cout<<"the  encryption message C is:"<<C<<endl;
        unsigned long p_long=PowMod(C, d, n);
        cout<<"the  decrypted message number is:"<<p_long<<endl;
        std::cout << "the  decrypted message P is:";
        vector<char> p_vector=strDecode(p_long);
    cout<<"=============output================"<<endl;
    printf("p: %lu,q: %lu,",p,q);
    cout<<"n: "<<n<<endl;
    cout<<"M:"<<M<<endl;
    cout<<"C:"<<C<<endl;
    cout<<"P:";
    strDecode(p_long);;

        return 0;
}
```